Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
FHWA/TX-07/0-5003-2		

4. Title and Subtitle	5. Report Date
TESTING FOR COMPLIANCE TO NTCIP STANDARDS	September 2006 Resubmitted: February 2007 Published: April 2007
	6. Performing Organization Code
7. Author(s) Robert De Roche	8. Performing Organization Report No. Report 0-5003-2
9. Performing Organization Name and Address Texas Transportation Institute The Texas A&M University System College Station, Texas 77843-3135	10. Work Unit No. (TRAIS) 11. Contract or Grant No. Project 0-5003
12. Sponsoring Agency Name and Address Texas Department of Transportation Research and Technology Implementation Office P.O. Box 5080 Austin, Texas 78763-5080	13. Type of Report and Period Covered Technical Report: September 2005-August 2006 14. Sponsoring Agency Code

15. Supplementary Notes

Project performed in cooperation with the Texas Department of Transportation and the Federal Highway Administration.

Project Title: Development of TxDOT Procedures and Specifications for Testing Device Compliance to NTCIP Standards

URL: http://tti.tamu.edu/document/0-5003-2.pdf

16. Abstract

The objectives of this two-year project are to define a framework for testing conformance to National Transportation Communications for ITS Protocol (NTCIP) standards, identify an approach to describe the extent to which testing is needed, and recommend the appropriate documentation for such testing activities. To meet the objectives, the first year's report included a summary of past and current efforts by various groups and organizations, a description of available testing tools, and the results of a survey undertaken to understand Texas Department of Transportation's (TxDOT's) testing process and needs. These topics were followed by discussions of the steps involved in conformance testing, how NTCIP requirements are specified, current TxDOT testing processes, reporting results, and the mapping of requirements to tests. The first year's report concluded with an enumerated list of recommendations to establish a testing framework.

This second year report looks at the details of testing documentation, provides estimates for developing test procedures for the various NTCIP-conformant field devices, discusses how to apply the procedures to the TxDOT testing processes, and presents an outline for training classes. The main portion of the report concludes with some additional recommendations to establish a testing framework. Appendices address modifications to TxDOT Closed Circuit Television (CCTV) specifications, a template for a TxDOT specification listing CCTV NTCIP requirements, a set of CCTV test procedures, test results reporting, miscellaneous communications test procedures, and a preliminary set of traffic signal controller procedures.

17. Key Words	18. Distribution Statement
NTCIP, ITS, Testing	No restrictions. This document is available to the public through NTIS: National Technical Information Service Springfield, Virginia 22161 http://www.ntis.gov

19. Security Classif.(of this report)	20. Security Classif.(of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	494	

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

TESTING FOR COMPLIANCE TO NTCIP STANDARDS

by

Robert De Roche
Senior Research Specialist
Texas Transportation Institute

Report 0-5003-2
Project 0-5003
Project Title: Development of TxDOT Procedures and Specifications for Testing Device Compliance to NTCIP Standards

Performed in cooperation with the
Texas Department of Transportation
and the
Federal Highway Administration

September 2006
Resubmitted: February 2007
Published: April 2007

TEXAS TRANSPORTATION INSTITUTE
The Texas A&M University System
College Station, Texas 77843-3135

DISCLAIMER

This research was performed in cooperation with the Texas Department of Transportation (TxDOT) and the Federal Highway Administration (FHWA). The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the official view or policies of the FHWA or TxDOT. This report does not constitute a standard, specification, or regulation. This report is not intended for construction, bidding, or permit purposes. The research supervisor in charge of the project was Robert De Roche.

The United States Government and the State of Texas do not endorse products or manufacturers. Trade or manufacturers' names appear herein solely because they are considered essential to the subject of the report.

ACKNOWLEDGMENTS

The research of this project was performed under a cooperative program between the Texas Transportation Institute (TTI), the Texas Department of Transportation, and the Federal Highway Administration. The author wishes to thank the Project Monitoring Committee. Ms. Carol Rawson served as the program coordinator, Mr. Fabian Kalapach served as the project director, and Messrs. Charlie Farnham, David Danz, and Steve Barnettt served as the project advisors. Mr. Wade Odell also served TxDOT research engineer.

Within TTI, the author wishes to thank Dr. Kevin Balke for reviewing and commenting on the report and Mr. Jeremy Johnson for his invaluable help in developing scripts.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

LIST OF ABBREVIATIONS AND SYMBOLS

AASHTO	American Association of State Highway and Transportation Officials
ASC	Actuated Signal Controller
ATMS	Advanced Transportation Management System
BER	Basic Encoding Rules
CCTV	Closed Circuit Television
CHAP	Challenge Handshake Authentication Protocol
CRC	Cyclic Redundancy Check
DMS	Dynamic Message Sign
DMS 11170-TSC	DMS-11170, Fully Actuated, Solid-State Traffic Signal Controller Assembly
DOT	Department of Transportation
DUT	Device Under Test
EIA	Electronics Industry Alliance
ELMS	Electrical and Lighting Management Systems
ESS	Environmental Sensor Station
FDOT	Florida Department of Transportation
FHWA	Federal Highway Administration
FMS	Field Management Station
FTP	File Transfer Protocol
HAR	Highway Advisory Radio
HDLC	High-Level Data Link Control
HITL	Hardware-in-the-Loop
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IPI	Initial Protocol Identifier
ISO	International Organization for Standardization
ITE	Institute of Transportation Engineers
ITL	Interoperability Test Lab
ITS	Intelligent Transportation Systems

MIB	Management Information Base
NEMA	National Electrical Manufacturers Association
NIATT	National Institute for Advanced Transportation Technology
NTCIP	National Transportation Communications for ITS Protocol
NTCIP 1103-TMP	NTCIP 1103 – Transportation Management Protocols
NTCIP 1201-GLO	NTCIP 1201 – Global Object Definitions
NTCIP 1202-ASC	NTCIP 1202 – Object Definitions for Actuated Traffic Signal Controller Units
NTCIP 1203-DMS	NTCIP 1203 – Object Definitions for Dynamic Message Signs (DMS)
NTCIP 1204-ESS	NTCIP 1204 – Environmental Sensor Station Interface Standard
NTCIP 1205-CCTV	NTCIP 1205 – Object Definitions for Closed Circuit Television (CCTV) Camera Control
NTCIP 1206-DCM	NTCIP 1206 – Object Definitions for Data Collection and Monitoring (DCM) Devices
NTCIP 1207-RMC	NTCIP 1207 – Object Definitions for Ramp Meter Control (RMC) Units
NTCIP 1208-SW	NTCIP 1208 – Object Definitions for Closed Circuit Television (CCTV) Switching
NTCIP 1209-TSS	NTCIP 1209 – Data Element Definitions for Transportation Sensor Systems
NTCIP 1210-FMS	NTCIP 1210 – Field Management Stations - Part 1: Object Definitions for Signal System Masters
NTCIP 1211-SCP	NTCIP 1211 – Object Definitions for Signal Control and Prioritization
NTCIP 1213-ELMS	NTCIP 1213 – Objects Definitions for Electrical and Lighting Management Systems
NTCIP 2101-PMPP/RS232	NTCIP 2101 – Point to Multi-Point Protocol Using RS-232 Subnetwork Profile
NTCIP 2102-PMPP/FSK	NTCIP 2102 – Point to Multi-Point Protocol Using FSK Modem Subnetwork Profile
NTCIP 2103-PPP	NTCIP 2103 – Point-to-Point Protocol over RS-232 Subnetwork Profile
NTCIP 2104-Ethernet	NTCIP 2104 – Ethernet Subnetwork Profile
NTCIP 2201-T2	NTCIP 2201 – Transportation Transport Profile

NTCIP 2202-ITP	NTCIP 2202 – Internet (TCP/IP and UDP/IP)Transport Profile
NTCIP 2301-STMF	NTCIP 2301 – Simple Transportation Management Framework Application Profile
NTCIP 8007-TEST	NTCIP 8007 – Testing and Conformity Assessment Documentation within NTCIP Standards Publications
NTCIP 9012-TG	NTCIP 9012 – Testing and Conformity Assessment User Guide for NTCIP Field Devices and Center-to-Field Communications
NTCIP Guide	NTCIP 9001 – The NTCIP Guide
OER	Octet Encoding Rules
OID	Object Identifier
PDF	Portable Document Format
PDU	Protocol Data Unit
PICS	Protocol / Profile Implementation Conformance Specification
PMPP	Point to Multi-Point Protocol
PPP	Point-to-Point Protocol
PRL	Protocol / Profile Requirements List
PTZ	Pan, Tilt, and Zoom
QPL	Qualified Products List
RFC	Request for Comment
RS	Recommended Standard
RTM	Requirements Traceability Matrix
SFMP	Simple Fixed Message Protocol
SNMP	Simple Network Management Protocol
SS 6025	TxDOT 2004 Special Specification 6025 CCTV Field Equipment
SS 6026	TxDOT 2004 Special Specification 6026 National Transportation Communications for ITS Protocol for Dynamic Message Signs
SS 6504	TxDOT 1993 Special Specifications 6504 – Testing, Training, Documentation and Warranty
STMP	Simple Transportation Management Protocol
TCL	Tool Command Language
TCP	Transmission Control Protocol

TERL	Traffic Engineering Research Lab
TFTP	Trivial File Transfer Protocol
TS	Traffic Section
TTI	Texas Transportation Institute
TxDOT	Texas Department of Transportation
UDP	User Datagram Protocol
UP	Unnumbered Poll
V&V	Validation and Verification
VIVDS	Video Imaging Vehicle Detection Systems

GLOSSARY

Compliance	Compliance is a condition that exists when an item meets all of the requirements of a procurement specification.
Conformance	Conformance is a condition that exists when an item meets all of the mandatory requirements as defined by a formal standard.
Management Application	A generic term for any computer-based software used to configure, control, or monitor the operation of a field device or devices.

CHAPTER 1: INTRODUCTION

PROJECT OBJECTIVES

The objectives of this two-year project are to define a framework for testing conformance to National Transportation Communications for Intelligent Transportation Systems Protocol (NTCIP) standards, identify the approaches used to describe the extent to which testing is needed, and recommend appropriate documentation for such testing activities. This research project will accomplish the following for TxDOT:

- Assist TxDOT in developing a comprehensive approach to testing Intelligent Transportation Systems (ITS) -related hardware and software to ensure conformance with national standards and compliance with TxDOT specifications.
- Identify TxDOT testing needs and available resources to meet those needs.
- Develop a framework, along with methodologies and procedures as needed, for conducting both laboratory and field-testing of devices.
- Assist TxDOT in evaluating options for testing of ITS hardware and software as part of procurement and construction projects.
- Assist TxDOT in developing procedures and reports for documenting the results of the testing program.
- Develop outlines for training courses that convey how to use and interpret the results of the testing program.

The goal of testing is to ensure that the design, implementation, and functionality of a product meet user needs and requirements. NTCIP standards define a set of protocols associated with communications technologies used in transportation-related products. These protocols ensure systems that integrate NTCIP-conformant products can communicate using a common language and describe information in a consistent manner. The goal of NTCIP testing is to ensure that a product follows the protocol rules that define the common language and that the information exchanged by the components of a system is meaningful and understood. The intent of this project is to look at tasks involved and methods used to check conformance to NTCIP. The intention is to look also at a means of integrating NTCIP testing into TxDOT's current testing program.

SCOPE OF PROJECT

The scope of this research is testing of conformance to NTCIP standards and compliance to TxDOT specifications that reference NTCIP standards. NTCIP standards define common methods and protocols that enable ITS devices to communicate. The standards also define the language and words used when communicating. In some cases, no other standards exist that define the meaning of the words or how the words relate to functionality; therefore, some NTCIP standards also define the functionality of an ITS device. This research looks at the types of testing involved in showing conformance to NTCIP standards and compliance to TxDOT specifications, the resources available to accomplish the tasks, the specific needs for testing by TxDOT, the current testing process within TxDOT, and the testing tools available to help in the testing process.

ORGANIZATION OF THIS REPORT

In addition to this introduction chapter, this report contains five chapters and seven appendices. Chapter 2 of this report discusses NTCIP testing documentation. Testing documentation not only includes actual test procedures but also documentation that relates to specifying NTCIP requirements and how testing verifies those requirements. Chapter 2 provides a description of steps involved in creating testing documentation. By way of example, discussion focuses on the full set documentation for CCTV camera controllers beginning with a description of the modifications to TxDOT Special Specification 6025 (SS 6025) to reference NTCIP requirements (*1*). Appendix A provides suggested changes to that specification. Since TxDOT uses a separate document to define NTCIP requirements, a discussion of the content for a new document titled NTCIP for CCTV Equipment ensues. This document cites what NTCIP standards, what mandatory and optional conformance groups within the standards, and what optional object definitions within the standards to reference. Appendix B contains a template for the new TxDOT specification.

Chapter 2 also deals with a Requirements Traceability Matrix (RTM) that correlates the requirements in TxDOT Special Specification 6025 to NTCIP Object Definitions and Test Procedure Identifiers and presents an introduction and general description of the CCTV test procedures. Appendix C of this report contains the actual procedures.

What follows is a discussion on reporting results. The researcher found that the clearest means of reporting results is to use the Protocol/Profile Requirements List (PRL) that appears in the NTCIP standards or the test procedure documentation. Appendix D contains a PRL that shows test results. The CCTV Test Results part of Chapter 2 shows a sample of how a test procedure can show test results.

The next section of Chapter 2 deals with traffic signal controller documentation. TxDOT department material specification DMS-11170, Fully Actuated, Solid-State Traffic Signal Controller Assembly (DMS 11170-TSC) contains NTCIP requirements *(2)*. As such, the researcher makes several suggestions for addressing communications. The next part discusses an initial RTM based upon the requirements in DMS 11170-TSC *(2)*. This is an initial RTM because it only deals with the requirements in DMS 11170-TSC, individual objects definitions from the relevant standards, and the limited number of traffic signal controller test procedures that exist *(2)*. Appendix E provides a description and listing of existing communications test procedures that may be applicable to any NTCIP field device. Appendix F contains a Test Design Specification for addressing all the functions of a traffic signal controller. It uses IEEE Std. 829 – IEEE Standard for Software Test Documentation as a guide on the organization and content *(3)*.

Chapter 3 provides estimates for the level of effort needed to develop test procedures for all NTCIP field devices. The estimates use statistics about the content of the NTCIP standards, the researcher's experience in developing test procedures, discussions with NTCIP working group chairs and editors, and current NTCIP development plans.

Chapter 4 deals with applying NTCIP testing procedures in TxDOT testing processes. Discussions deal with TxDOT's internal process consisting of Qualified Products List (QPL), sample environmental, configuration, and system testing. It also deals with TxDOT's external process or contractor testing consisting of design approval, demonstration, stand-alone, and system integration test steps. The discussions focus on how NTCIP testing might apply to each of these processes and steps. Chapter 4 concludes with a brief discussion of configuration management and version control.

Chapter 5 presents two training course outlines. The first looks at testing from an NTCIP perspective, and the second outline examines testing from a TxDOT perspective. The first explores the difference between conformance and compliance testing. Conformance testing applies to the NTCIP standards, and compliance testing applies to TxDOT specifications. The outline suggests that the training first look at background information and the two types of standards: data dictionaries and protocols/profiles. It would then cover how to view the NTCIP framework in order to understand how individual standards make up an actual implementation. Topics on terminology and techniques and on how best to interpret and report the results follow.

The second outline in Chapter 5 looks at testing from the TxDOT perspective. It deals with the how to balance a desire to fully test an implementation with the reality issues of lack of resources and time. A discussion of risk management looks at how to minimize the testing effort and still maintain a high level of confidence. Further topics cover what to test, the techniques to use, and the various testing tools that are available. The chapter concludes with an outline of how to address configuration management and includes a suggested evaluation form.

Chapter 6 of the report adds some additional recommendations on defining a framework for the testing of conformance to NTCIP and integrating it into the current TxDOT testing program. The recommendations are in addition to the 17 recommendations that the first year report included.

CHAPTER 2:
NTCIP TESTING DOCUMENTATION

INTRODUCTION

A major component of any testing framework is supporting documentation. There must be specific requirements or specifications, a document that correlates those requirements to a set of test procedures for verifying the requirements, and documented test procedures to ensure that the verification process is consistent. For any organization as large as TxDOT, consistent reporting of results should also be a part of the framework.

By specifying conformance to NTCIP standards, one can assume that TxDOT shares the view the standards promote:

- Compatibility
- Interoperability
- Interchangeability

System components should be compatible so that different components can share a common communications infrastructure. There should also be interoperability between system components so that components from different vendors can work together to provide the necessary functionality. System components should also exhibit interoperability so that a system component from one vendor can replace that of another without any change in functionality.

In the past, ITS field devices did not exhibit compatibility to any significant degree. Most manufacturers used their own proprietary protocols to communicate, which made sharing of a communications link by multiple manufacturers impossible. Management applications had to develop drivers for each brand and type of device. Some projects did achieve a sense of compatibility by mandating support for a system integrator’s protocol such as Protocol 90 or Management Information System for Traffic (MIST). These protocols, however, are for use with traffic signal controllers and not other ITS field devices. At the communications infrastructure level, NTCIP protocols are device independent.

To achieve interoperability, the effort to standardize communications involved three standards organizations: the American Association of State Highway and Transportation Officials (AASHTO), the Institute of Transportation Engineers (ITE), and the National Electrical Manufacturers Association (NEMA). These three organizations brought together public sector representatives, consultants, and equipment manufacturers to work out a common language for expressing information and sets of messages to exchange the information. By agreeing upon these common protocols, management applications can send information to a type of field device and have it understood by the device, as well as, receive information from the device and understand what it is saying.

By defining the meaning of the words in the common language and their effect upon a device, the NTCIP standards achieve interchangeability. The standards establish a minimum level of common functionally. While a system and its components are free to go beyond the common functionality, subscribing to the NTCIP standards ensures that a level of interchangeability is always present.

For TxDOT to subscribe to NTCIP means that its specifications must specify requirements for NTCIP standards. So far, TxDOT has a start in that direction in a number of areas. After reviewing ITS field device specifications that appear on the TxDOT Expressway webpage, the researcher found that the statewide use specifications for traffic signal controllers and dynamic message signs have fully detailed requirements (*4*). Similar specifications for video imaging vehicle detection systems (VIVDS) and spread spectrum radios also cite NTCIP requirements, but only in general, with no details of what that entails. Other devices such as on-street masters, closed circuit television, electrical lighting and management systems, and ramp metering controller do not have any NTCIP requirements. This research further discusses that transition by looking at the documentation to specify NTCIP requirements and then developing test procedures to test for compliance to the NTCIP requirements.

CCTV DOCUMENTATION

This section looks at the documentation to add NTCIP requirements to TxDOT's CCTV specifications and test procedures to verify those requirements. Special Specification 6025 - CCTV Field Equipment does not currently address NTCIP requirements (*1*). Suggested modifications to that specification appear in Appendix A. The first part of this section presents the reasoning behind those changes.

Before one can properly address specific NTCIP requirements, one needs to correlate the general requirements (in SS 6025) to the object definitions and functionality in NTCIP standards (*1*). The recommended method of ensuring that test procedures meet user needs and requirements is through the use of an RTM. Text and examples illustrate the composition and elements of an RTM. The text elaborates on the steps and process of creating an RTM. Table 3 then presents a fully developed RTM for SS 6025 (*1*). While it simplifies the expression of user needs, the RTM does map them to requirements and references in SS 6025, cites the relevant NTCIP objects, and correlates these to specific test procedure identifiers. The test procedure identifiers relate to a set of CCTV test procedures that appear in Appendix C.

With the NTCIP object definitions and functionality identified, one can proceed to develop test cases and test procedures to ensure that a device implements the definitions and functionality correctly. In some cases, this may be a simple process of reviewing test procedures that are already in the public domain. In other cases, one may need to consider a formal test plan.

The first case applies to CCTV. A number of test procedures and test cases already exist. A review of the procedures and test cases indicates where they apply. Assigning them test case numbers and references allows a mapping to the RTM. In the case of the documentation for traffic signal controllers, the process was the development of internal documentation that sets out a plan, identifies test cases, and outlines individual test cases.

The last part of the testing documentation to consider is the presentation of the testing results. While most testing tools produce some type of test report, the researcher believes that such a report should include how a device was tested. The researcher presents an approach that uses documentation that may appear in the NTCIP standards.

Special Specification CCTV Field Equipment

The current special TxDOT specification for CCTV equipment does not address NTCIP. Appendix A is a modified version of the SS 6025 (*1*). The text written in italics or shown crossed out illustrates modifications that come from other special specifications that include a reference to NTCIP requirements.

The first modification is an additional sentence in the description clause, which is a reference to another special specification dealing with the specifics of NTCIP requirements. The typical wording that appears in other special specifications is:

> *The following special specifications is referenced in this specification: "National Transportation Communications for ITS Protocol for CCTV Equipment"*

The next modification deals with the clause describing the communications interface. The exact wording may differ in particular special specifications. For SS 6025, however, the chances consist of referencing the Special Specifications NTCIP for CCTV Equipment, removal of optional data rates that are in conflict with NTCIP, and removal of any functional requirements related to the protocol used (*1*).

Another change is the reference to "EIA-232 C/D port." The term "RS (Recommended Standard)-232 Serial port" generalizes the requirement so that specifics come from Special Specifications NTCIP for CCTV Equipment. The change uses the term "RS" in place of "EIA" because the formal definition of the 9-pin version of the RS-232 interface comes from EIA (Electronics Industry Alliance)-574. The change drops the term C/D because the NTCIP standards call for a newer version of EIA-232 standard (Version F).

Initial Requirements Traceability Matrix

Once there is a specification that defines the functional requirements, the next step is to develop an RTM that maps the requirements to NTCIP object definitions. The purpose of the RTM is to help ensure the functionality of the requirements maps to one or more NTCIP objects that either define, control, or provide status related to that function. This ensures that NTCIP conforms to the requirements and that requirements have NTCIP support.

A traceability matrix verifies that all stated and derived requirements are allocated to at least one NTCIP object (forward tracing). The matrix lists the source of requirements (backward tracing). While the RTM in this document focuses on NTCIP, a matrix can include tracing to things other than NTCIP objects such as capabilities, physical requirements, test procedures, etc. For example, a user need could be to operate outdoors and require equipment over a certain temperature range.

Table 1 is the start of an RTM that references NTCIP objects. Table 3 is a fully expanded RTM that adds references to test procedures.

The procedure for creating an RTM consists of:

- identifying user needs (nouns) that correspond to the names of functions,
- referencing the text that describes the action (verb) or usage of the user need,
- providing a cross-reference to an item or clause of the requirements document, and
- correlating the user need to the NTCIP standard or object that supports the need.

Table 1. Requirements Traceability Matrix with NTCIP Objects.

SS 6025 Closed Circuit Television NTCIP Requirements Traceability Matrix			
User Need	**Requirement**	**TxDOT SS 6025 Reference**	**NTCIP Object Support**
Traceable to NTCIP 1205-CCTV			
Provide Remote Control			
Shutter Speed	Provide remotely selectable shutter speed	2.B.1	None
Zoom	Provide a lens with capabilities for remote control of zoom operations	2.B.2	3.2.8 rangeZoomLimit 3.3.3 timeoutZoom 3.5.3 positionZoomLens 3.5.8 positionQueryZoom
Long-Term Exposure	Provide control receivers for Digital Signal Processing (DSP) of long-term exposure control	6.	None
Auto-Focus	Provide control receivers for Digital Signal Processing of auto-focus control	6.	3.2.9 rangeFocusLimit 3.3.4 timeoutFocus 3.6.4 systemLensFeatureControl 3.6.5 systemLensFeatureStatus 3.6.6 systemLensEquipped
Auto/Manual Focus Control	Provide units with control receivers for DSP Function – auto/manual focus control	6.	3.2.9 rangeFocusLimit 3.3.4 timeoutFocus 3.6.4 systemLensFeatureControl 3.6.5 systemLensFeatureStatus 3.6.6 systemLensEquipped

In the case of conformance and compliance related to NTCIP, the list of user needs is limited to parameters, controls, and status information that relate to communications. For CCTV, the needs primarily come from remote control. Any quantities, limits, or values (constraints) that apply to the needs should also appear in the list. Examples of constraints are number of presets, tilt limits, and label color choices. The next column in the RTM is for stating the requirement with some action. This typically consists of a short sentence putting the user need in context. For example, if the need is a zoom capability, then stating that zoom has to be controlled remotely puts it in the context of NTCIP. The third column lists the reference to the item, clause, or heading number in the specification from which user need and requirement is derived. This provides the backward traceability.

To complete the RTM for this stage, one enters the object names from NTCIP Standards that relate to user and requirement. The NTCIP object names usually include one or more of the same words used to define the user needs. The NTCIP object description field may also contain a reference. It is typical to cite the same NTCIP objects in many places. At this point, the RTM should look similar to Table 1.

A number of NTCIP standards now include RTM information. Table 2 is part of an RTM that appears in NTCIP 1203 – Object Definitions for Dynamic Message Signs (DMS) (NTCIP 1203-DMS) (*5*). While the RTM in that standard includes additional columns of information, the functional requirement and object names that support the requirement are the essential items. Even if an NTCIP standard includes an RTM, it is advisable to generate one based upon the requirements defined in TxDOT specifications. The need to do this becomes apparent when one realizes that the NTCIP standards may not provide support for all functional requirements in a specification. NTCIP standards also may require some functionality that does not correspond to a requirement in the specification. For example, the functional requirement to provide remotely selectable shutter speed or control of long-term exposure in SS 6025 does not have object support in NTCIP 1205 – Object Definitions for Closed Circuit Television (CCTV) Camera Control (NTCIP 1205-CCTV) (*1*,*6*). NTCIP 1205-CCTV has support for remote on-screen menu control that does not appear in SS 6025 (*6*,*1*).

Table 2. Requirements Traceability Matrix from NTCIP 1203-DMS.

FR ID	Functional Requirement	Interface ID	Interface	Dialog ID	Object ID	Object
D.3.1.1	Determine Device Component Information			D.4.2.1		
		D.4.3.4.1	Module Table			
					D.2.2	globalMaxModules
		D.4.3.4.2	Module			
					D.2.3.1	moduleNumber
					D.2.3.2	moduleDeviceNode
					D.2.3.3	moduleMake
					D.2.3.4	moduleModel
					D.2.3.5	moduleVersion
					D.2.3.6	moduleType

After filling out the NTCIP object support column (see Table 1), the list of object names will identify specific objects and conformance groups that an implementation needs to support and suggest what a potential test procedure needs to address. The list will also indirectly identify objects and requirements that may not be in the requirement specification. After generating an RTM based solely upon the user needs and requirements in SS 6025, the following is the list of objects or groups in NTCIP 1205-CCTV v01.10 that are traceable to a requirement defined in SS 6025 (*1*,*6*):

- 3.2.1 rangeMaximumPreset
- 3.2.2 rangePanLeftLimit
- 3.2.3 rangePanRangeLimit
- 3.2.4 rangePanHomePosition
- 3.2.6 rangeTiltUpLimit
- 3.2.7 rangeTiltDownLimit
- 3.2.8 rangeZoomLimit
- 3.2.9 rangeFocusLimit
- 3.2.10 rangeIrisLimit
- 3.2.11 rangeMinimumPanStepAngle
- 3.2.12 rangeMinimumTiltStepAngle
- 3.3.1 timeoutPan
- 3.3.2 timeoutTilt
- 3.3.3 timeoutZoom
- 3.3.4 timeoutFocus
- 3.3.5 timeoutIris
- 3.4.1 presetGotoPosition
- 3.4.2 presetStorePosition
- 3.4.3 presetPositionQuery
- 3.5.1 positionPan
- 3.5.2 positionTilt
- 3.5.3 positionZoomLens

- 3.5.4 positionFocusLens
- 3.5.5 positionIrisLens
- 3.5.6 positionQueryPan
- 3.5.7 positionQueryTilt
- 3.5.8 positionQueryZoom
- 3.5.9 positionQueryFocus
- 3.5.10 positionQueryIris
- 3.6.1 systemCameraFeatureControl
- 3.6.2 systemCameraFeatureStatus
- 3.6.3 systemCameraEquipped
- 3.6.4 systemLensFeatureControl
- 3.6.5 systemLensFeatureStatus
- 3.6.6 systemLensEquipped
- 3.7 CCTV Alarm Objects
- 3.9 CCTV Discrete Output Objects
- 3.11 CCTV Label Objects

Comparing the above list to the heading numbers and object names in NTCIP 1205-CCTV, the following is a list of object names and groups in NTCIP 1205-CCTV v01.10 that do not trace to a requirement defined in SS 6025 (*6*,*1*):

- 3.2.5 rangeTrueNorthOffset
- 3.8 CCTV Discrete Input Objects
- 3.10 CCTV Zone Objects
- 3.12 CCTV On-Screen Menu Control Objects

At this point, one may want to consider adding a user need or functional requirement in the specification to address the functionality expressed by the objects. If one determines that they are not necessary, the special specification that defines NTCIP for the specific equipment need not require them even if the NTCIP standard indicates that they are mandatory. This gets at the heart of the distinct difference between conformance to the NTCIP and compliance with a TxDOT specification. To show conformance to a NTCIP standard, an implementation needs to support mandatory objects and have tests run to determine if it is correct. To show compliance with TxDOT specifications, an implementation does not need to support objects that have no functional requirement, and one does not need to run tests on those objects.

One can also use an RTM to summarize user needs and requirements that have no support within the NTCIP standard. These non-supported items would show up as a blank in the NTCIP object support column of the RTM. The following requirements in TxDOT Special Specification 6025 do not trace to an object name or group in NTCIP 1205-CCTV v01.10 (*1*,*6*):

- Shutter Speed
- Long-Term Exposure
- Remote White Balancing Control
- Auto and Manual White Balance Control

After identifying the unsupported needs and requirements, one may want to consider whether they are truly required. If so, a manufacturer may support them through proprietary objects. The manufacturer may also have a set of procedures that provide functional testing.

The next step is to determine if the TxDOT specification needs to address additional objects. The following is the list of object names or groups referenced in NTCIP 1205 – Object Definitions for Closed Circuit Television (CCTV) Camera Control (NTCIP 1205-CCTV) v01.10 (and defined in NTCIP 1201 – Global Object Definitions [NTCIP 1201-GLO]) that do not trace to a requirement defined in SS 6025 (*6*,*7*,*1*):

- B.7 Global Configuration Conformance Objects
- B.8 Security Conformance Group

NTCIP 1205-CCTV does not address application, transport, and subnetwork-level requirements (6). Even though the object definitions use the Simple Network Management Protocol (SNMP) macro to describe them, there are several choices for encoding information and exchanging it over different media. A discussion of these choices appears in the next section. However, there may be objects and conformance groups that relate to the setup, control, and status at the application, transport, or subnetwork-levels. Special specifications related to NTCIP that include references to specific application, transport, and subnetwork-level requirements may not be valid for statewide use. For example, one project may require an RS-232 interface while another may require an Ethernet interface. The objects and conformance groups would be different.

Based upon the general description of the communications protocols appearing in SS 6025, the assumption is that the application-level protocol is SNMP as defined in NTCIP 2301 – Simple Transportation Management Framework Application Profile (NTCIP 2301-STMF) (*1*,*8*). The following is the list of object names or groups referenced in NTCIP 2301:2001 that do not trace to a requirement defined in SS 6025 (*1*,*8*):

- A.7.1.1 System Group
- A.7.1.2 SNMP Group
- A.5.4 SNMP Configuration Conformance Group

Since the communications protocol description does not mention a networking capability, the assumption is that the transport-level protocol is the Null protocol as defined in NTCIP 2201 – Transportation Transport Profile (NTCIP 2201-T2). There are no objects or groups called for in this standard (*9*).

From SS 6025's description, the assumption is that the subnetwork-level protocol is the point-to-multi-point protocol (PMPP) as defined in NTCIP 2101 – Point to Multi-Point Protocol Using RS-232 Subnetwork Profile (NTCIP 2101-PMPP/RS232) (*1*,*10*). The following is the list of object names or groups referenced in NTCIP 2101-PMPP/RS232 that do not trace to a requirement defined in SS 6025 (*10*,*1*):

- A.7.1 HDLC (High-Level Data Link Control) Group
- A.7.2 RS232 Asynchronous Group
- A.7.3 HDLC Group Address Group

With the base specification put in the form of an RTM one can then address the language that goes into a specific specification dealing with NTCIP related requirements. The completed RTM (less the test procedure column) shown in Table 3 (see page 21) serves as the basis for a specification for NTCIP for CCTV equipment.

Special Specification NTCIP for CCTV Equipment

Since TxDOT does not have a document that addresses NTCIP requirements related to CCTV Equipment, Appendix B is the researcher's suggestion for a TxDOT special specification that deals with the particulars. Using TxDOT 2004 Specifications - Special Specification 6026 – National Transportation Communications for ITS Protocol for Dynamic Message Signs (SS 6026) as a model for format and wording, the document in Appendix B covers the communications protocol and data dictionary (object definitions) requirements that are specific to CCTV equipment. Appendix B serves the purpose of discussion and provides suggested wording (*11*).

In Appendix B, Item 1 - Description is typical boilerplate wording that TxDOT uses in similar documents. Item 2 - Requirements begins with two paragraphs of boilerplate information.

Items 2.A through 2.G deal with specific NTCIP requirements as they relate to CCTV equipment. Items 2.H through 2.J cover TxDOT specific values and ranges, an operational requirement, and documentation requirements. Item 3- Testing and Verification and Item 4 - Measurement and Payment are typical boilerplate information, as well.

As in SS 6026, items 2.A though 2.G of Appendix B cover the communications protocol and profile requirements *(11)*. Functional requirements dictate the choices but NTCIP 9001 – The NTCIP Guide (NTCIP Guide) provides a framework for how all the various protocols and profiles fit together *(12)*. As stated in the NTCIP Guide,

> *"To ensure a working system, deployers must specify and/or select an NTCIP protocol or profile at each level."*

Item 2.A defines the subnetwork level profile requirements. Since the original SS 6025 references an interface through an EIA-232 C/D port, the NTCIP 2101-PMPP/RS232 standard is now the reference (*1*,*10*). This is the standard that most CCTV manufacturers support. TxDOT should understand that this standard might not be appropriate to all CCTV implementations. Some may warrant a dial-up (NTCIP 2103 – Point-to-Point Protocol over RS-232 Subnetwork Profile [NTCIP 2103-PPP]) or Ethernet (NTCIP 2104 – Ethernet Subnetwork Profile [NTCIP 2104-Ethernet]) subnetwork (*13*,*18*).

Item 2.B defines the transport level profile requirements. Since the original SS 6025 does not mention any networking capability, NTCIP 2201-T2 serves as the reference *(9)*. Most CCTV manufacturers support this standard. NTCIP 2201-T2 addresses a multiplexing capability by defining various parsing and encapsulation methods *(9)*. The multiplexing scheme would permit several different protocols to exist at the application level. Since CCTV devices do not appear to need this functionality, the requirement only references parsing method 1 and encapsulation method 1.

Item 2.C defines the application level profile requirements. The reference is to NTCIP 2301-STMF *(8)*. This standard defines the use of SNMP as the industry standard protocol for encoding data and exchanging the data between a management application and an ITS field device. NTCIP 2301-STMF actually defines three different protocols: SNMP, Simple Transportation Management Protocol (STMP), and Simple Fixed Message Protocol (SFMP) *(8)*.

SNMP offers simplicity and flexibility but adds overhead with its encoding method. STMP offers flexibility and compact encoding but is not simple to set up and use. SFMP offers simplicity and compact encoding but does so by being less flexible as to what kind of information it handles. Given the amount of information in the exchanges and the frequency of the exchanges, Conformance Level 1 of NTCIP 2301-STMF (only the SNMP portion) is the appropriate choice *(8)*. Most CCTV manufacturers support this conformance level.

Items 2.D through 2.H define the information level profile requirements. These items cover:

- basic configuration and control object definitions/data elements of a CCTV,
- two optional objects that relate to the position of the camera,
- two SNMP-related groups that identify the equipment and provide communications troubleshooting information, and
- three PMPP-related configuration and communications troubleshooting information groups.

Item 2.D identifies the mandatory and optional conformance groups with which the CCTV equipment must comply to meet TxDOT specifications. These consist of:

- CCTV Configuration – covering limits, timeouts, and labels
- Extended Functions – covering features such as power, heater, focus, alarms, and I/O
- Motion Control – covering presets and positioning controls
- Configuration – covering manufacturer make and model information

The following is the list of objects names and conformance groups within NTCIP 1205-CCTV that do not correspond to a requirement in SS 6025 and therefore have no reference in Item 2.D (6,1):

- CCTV Discrete Input Objects (clause 3.8 of NTCIP 1205-CCTV)
- CCTV Zone Objects (clause 3.10 of NTCIP 1205-CCTV)
- CCTV On-Screen Menu Control Objects (clause 3.12 of NTCIP 1205-CCTV)

Item 2.E identifies two optional objects from the Motion Control Conformance Group and another object from the Configuration Conformance Group. These three objects do not have a specific requirement in SS 6025 but may be useful *(1)*. The objects positionQueryFocus and positionQueryIris can provide a user with current position information about focus and iris. The globalSetIDParameter indicates any changes have been made to parameter type objects (objects whose value is retained even if power is lost).

Item 2.F identifies three conformance groups that come from NTCIP 2301-STMF (*8*). The first group is the System Conformance Group. This group is a set of generic object definitions from Request for Comment (RFC) 1213 - Management Information Base for Network Management of TCP/IP-based internets: MIB-II (RFC 1213) that apply to any device implementing SNMP (*14*). They provide system identification, contact information, and other general information about the equipment. The second group is SNMP Statistics. This group is another set from RFC 1213 that provides counters that keep track of the number of messages received, how many different types of messages there are, the number of errors, how many messages had the wrong community name, and other counters related to SNMP operations (*14*). The last group is a single object in the SNMP Configuration Conformance Group. The object, snmp-maxPacketSize, defines how many bytes long a message may be at the application level.

Item 2.G defines three conformance groups that are mandatory requirements for devices that are conformant to NTCIP 2101-PMPP/RS232 and support SNMP. The first group is the HDLC Conformance Group (*10*). This group consists of two timers that control response timing, two control parameters that define how many bytes long a message may be at the subnetwork-level, and an identifier of the port number to which the parameters apply. The second group is the RS232 Asynchronous Group. This group defines the port type and data rate at which the interface operates. The group also has counters that keep track of any framing and overrun errors that may occur. The last group is HDLC Group Address. This group defines a number of addresses that a device would listen to for broadcast messages. For example, setting the time-of-day is one example that might be a broadcast message.

Items 2.F and 2.G are information-level object definitions that are mandatory to support if NTCIP 2301-STMF and NTCIP 2101-PMPP/RS232 are applicable (*8,10*). They are mandatory if a CCTV Equipment is conformant to the NTCIP standards. For compliance to TxDOT specifications, the researcher does not believe that HDLC Group Address Conformance Group is a requirement.

TxDOT should consider whether all or parts of the SNMP Statistics, SNMP Configuration, and HDLC Conformance Group require support. The researcher believes that SNMP statistics would be helpful in analyzing the amount and type of messages and useful in troubleshooting communications problems. However, their utility in a non-networked environment is questionable. Entries in the Minimum Project Requirements table could replace snmp-maxPacketSize in SNMP Configuration and several objects in the HDLC Group. From a system's perspective, if other types of devices use the same application and subnetwork level profiles, the object support should be the same across all of them. For CCTV equipment, however, they are not essential.

Items 2.A through 2.G cover the four profile/protocol levels. To ensure that a specification identifies and addresses each level, the researcher recommends that item numbers specifically identify each level. For example, adding the following item numbers would clearly indicate and identify the requirements at each level:

- 2.1 Subnetwork-level requirements
- 2.2 Transport-level requirements
- 2.3 Application-level requirements
- 2.4 Information-level requirements
- 2.4 Project requirements and documentation

Item 2.H covers Minimum Project Requirements. The intent of NTCIP is to provide communication and information standards that are applicable to the transportation industry as a whole. The SYNTAX values that appear in numerous objects are more for purposes of understanding the encoding size rather than any state or project specific values. The technique to add state or project specific variances is to subrange the values. The typical means of expressing the variances is through Minimum Project Requirements. The following is a discussion of the value of each object in the table.

The minimum value of labelMaximum is 16. The object represents the minimum number of labels that the CCTV equipment must support. SS 6025 does not define a value so this is an arbitrary choice (*1*). One manufacturer supports 64.

The value for labelColor is white. NTCIP 1205-CCTV specifies this as the default color that is mandatory to support. SS 6025 indicates that text should be white letters and black outline (*6*,*1*).

The value of rangeMaximumPreset is 16. The object defines the minimum number of preset positions that CCTV equipment must support. Presets include pan, tilt, zoom, and focus values. SS 6025 does not define a value so this is an arbitrary choice (*1*). One manufacturer supports 64.

The values for rangePanLeftLimit and rangePanRightLimit are both set to 35999. The objects define the maximum angle that a camera can pan from the home position. SS 6025 specifies a horizontal movement of 360° full, contiguous rotation movement (*1*). If the requirement is to have the CCTV continuously to the right or left without any limit, then this value should be 65535.

The value of rangePanHomePosition is set to 0. The object defines an arbitrary point on a circle from which to measure the left and right limits. The value of rangeTrueNorthOffset is set to 65535. This means that there is no support for a true North offset from the home position. SS 6025 does not specify equivalent requirements (*1*).

The researcher has set the minimum requirement for rangeTiltUpLimit and rangeTiltDownLimit to +4000 and -9000, respectively. These values correspond to the values in SS 6025, 40 degrees up and 90 degrees down (*1*). In looking at several special specifications for district use, the requirements ranged from 20 to 90 degrees up and from 90 to 110 degrees down.

The researcher has set the minimum requirement for rangeZoomLimit to 65535. This corresponds to a scalar focus positioning value of telephoto. The values for rangeFocusLimit and rangeIrisLimit are set to 0. A 0 value corresponds to non-support of the limits. The values for rangeMimimumPanStepAngle and rangeMimimumTiltStepAngle are set at 10. This corresponds to value of 0.1 degrees. SS 6025 does not define equivalent requirements (*1*). The value is somewhat arbitrary but is one manufacturer's supported value.

The value of systemCameraEquipped is set to 128 to indicate that the CCTV is only to have controllable camera power. NTCIP 1205-CCTV allows for a controllable heater, wiper, washer, and blower (*6*). However, SS 6025 does not list these as requirements and, therefore, the value represents the required support (*1*).

The value of systemLensEquipped is set to 172 to indicate that the CCTV is to have controllable Auto Iris and Auto Focus. SS 6025 defines the Auto Iris requirement in Item 2 and the Auto Focus requirement in Item 6 (*1*).

The values for zoneMaximum, zoneCameraEquipped, and menuControl are set to N/A (not applicable). SS 6025 does not require zones or on-screen menu control features (*1*).

The definition of the last two items in the Minimum Project Requirements table comes from NTCIP 1201-GLO (*7*). The communityNamesMax value is set to 3. This corresponds to an administrator having the ability to define three different community names (users) that could allow different access rights to the information in a CCTV. For example, one user could have full read-write access while another could be limited to just read-only access. It is manufacturer specific how communityNameAccessMask would be set to do this.

Returning to discussion of the numbered items of Item 2, Item 2.I – Hardware Limitations relates a general functional requirement of how SNMP operates. Since NTCIP 2301-STMF covers this, the researcher does not feel its inclusion is necessary (*8*).

Item 2.J – Documentation discusses the requirements for supplying Management Information Bases (MIBs). In the first year's report on this project, the researcher recommended that TxDOT create or define MIBs that reflect TxDOT specifications. This MIB should be the one that a management application uses and the one used to perform any testing. One reason for doing this is that changes in versions of any standards can introduce incompatibilities. For example, NTCIP 1201-GLO Version 2 inadvertently made a change that introduced an incompatibility with implementations built to Version 1 (*7*). If the testing of implementations of Version 1 used the MIB defined in Version 1, there was no problem. If the testing of implementations of Version 2 used the MIB defined in Version 2, there was no problem. The problem was uncovered only when the versions were crossed.

Expanded Requirements Traceability Matrix

Once the specification of NTCIP for specific equipment is defined, test procedures that verify that an implementation meets the specification are created and cross-referenced in an RTM. Table 3 is the completed RTM.

Table 3. Requirements Traceability Matrix with Test Procedures.

SS 6025 Closed Circuit Television Requirements Traceability Matrix				
User Need	**Requirement**	**TxDOT SS 6025 Reference**	**NTCIP Object Support**	**Test Procedure Identifier**
Traceable to NTCIP 1205-CCTV				
Provide Remote Control				
Shutter Speed	Provide remotely selectable shutter speed	2.B.1	None	None
Zoom	Provide a lens with capabilities for remote control of zoom operations	2.B.2	3.2.8 rangeZoomLimit 3.3.3 timeoutZoom 3.5.3 positionZoomLens 3.5.8 positionQueryZoom	Config-TC006 Zoom-TC003 Zoom-TC001 Zoom-TC002
Long-Term Exposure	Provide control receivers for Digital Signal Processing of long-term exposure control	2.B.6	None	None
Auto-Focus	Provide control receivers for Digital Signal Processing function of auto-focus control	2.B.6	3.2.9 rangeFocusLimit 3.3.4 timeoutFocus 3.6.4 systemLensFeatureControl 3.6.5 systemLensFeatureStatus 3.6.6 systemLensEquipped	Config-TC007 Focus-TC003 Lens-TC002 and Lens-TC003 Lens-TC002 and Lens-TC003 Lens-TC001, Lens-TC002, and Lens-TC003
Auto/Manual Focus Control	Provide units with control receivers for DSP function of auto/manual focus control	2.B.6	3.6.4 systemLensFeatureControl 3.6.5 systemLensFeatureStatus 3.6.6 systemLensEquipped	Lens-TC002 and Lens-TC003 Lens-TC002 and Lens-TC003 Lens-TC001, Lens-TC002, and Lens-TC003
I.D. Generator Operation	Provide units with control receivers for DSP function of I.D. generator operation	2.B.6	3.11 CCTV Label Objects	Label-TC001 and Label-TC002

Table 3. Requirements Traceability Matrix with Test Procedures (continued).

SS 6025 Closed Circuit Television Requirements Traceability Matrix				
User Need	**Requirement**	**TxDOT SS 6025 Reference**	**NTCIP Object Support**	**Test Procedure Identifier**
Alarm Function Control	Provide units with control receivers for DSP function of alarm function control	2.B.6	3.7 CCTV Alarm Objects	Alarm-TC001, Alarm-TC002, Alarm-TC003, Alarm-TC004, Alarm-TC005, Alarm-TC006, and Alarm-TC007,
Pan/Tilt Position preset	Provide units with control receivers for DSP function of pan/tilt position preset	2.B.6	3.4.1 presetGotoPosition 3.4.2 presetStorePosition 3.4.3 presetPositionQuery	Zone-TC001 Zone-TC001 Zone-TC001
Pan Left	Provide units with control receivers for DSP function of pan left	2.B.6	3.2.2 rangePanLeftLimit 3.2.4 rangePanHomePosition 3.2.11 rangeMinimumPanStepAngle 3.3.1 timeoutPan 3.5.1 positionPan 3.5.6 positionQueryPan	Config-TC003 Config-TC011 and Pan-TC002 Config-TC009 Pan-TC003 Pan-TC001, Pan-TC002, Pan-TC003, and Pan-TC004 Pan-TC002

Table 3. Requirements Traceability Matrix with Test Procedures (continued).

SS 6025 Closed Circuit Television Requirements Traceability Matrix				
User Need	**Requirement**	**TxDOT SS 6025 Reference**	**NTCIP Object Support**	**Test Procedure Identifier**
Pan Right	Provide units with control receivers for DSP function of pan right	2.B.6	3.2.3 rangePanRightLimit 3.2.4 rangePanHomePosition 3.2.11 rangeMinimumPanStepAngle 3.3.1 timeoutPan 3.5.1 positionPan 3.5.6 positionQueryPan	Config-TC003 Config-TC011 and Pan-TC002 Config-TC009 Pan-TC003 Pan-TC001, Pan-TC002, Pan-TC003, and Pan-TC004 Pan-TC002
Tilt Up	Provide units with control receivers for DSP function of tilt up	2.B.6	3.2.6 rangeTiltUpLimit 3.2.12 rangeMinimumTiltStepAngle 3.3.2 timeoutTilt 3.5.2 positionTilt 3.5.7 positionQueryTilt	Config-TC005 Config-TC009 Tilt-TC003 Tilt-TC001, Tilt-TC002, Tilt-TC003, and Tilt-TC004 Tilt-TC002
Tilt Down	Provide units with control receivers for DSP function of tilt down	2.B.6	3.2.7 rangeTiltDownLimit 3.2.12 rangeMinimumTiltStepAngle 3.3.2 timeoutTilt 3.5.2 positionTilt 3.5.7 positionQueryTilt	Config-TC005 Config-TC009 Tilt-TC003 Tilt-TC001, Tilt-TC002, Tilt-TC003, and Tilt-TC004 Tilt-TC002

Table 3. Requirements Traceability Matrix with Test Procedures (continued).

SS 6025 Closed Circuit Television Requirements Traceability Matrix				
User Need	**Requirement**	**TxDOT SS 6025 Reference**	**NTCIP Object Support**	**Test Procedure Identifier**
Zoom and Focus Position Preset	Provide units with control receivers for DSP function of zoom and focus position preset	2.B.6	3.4.1 presetGotoPosition 3.4.2 presetStorePosition 3.4.3 presetPositionQuery	Zone-TC001 Zone-TC001 Zone-TC001
Zoom In	Provide units with control receivers for DSP function of zoom in	2.B.6	3.2.8 rangeZoomLimit 3.3.3 timeoutZoom 3.5.3 positionZoomLens 3.5.8 positionQueryZoom	Config-TC006 Zoom-TC0003 Zoom-TC001, Zoom-TC002, Zoom-TC003, and Zoom-TC004 Zoom-TC002
Zoom Out	Provide units with control receivers for DSP function of zoom out	2.B.6	3.2.8 rangeZoomLimit 3.3.3 timeoutZoom 3.5.3 positionZoomLens 3.5.8 positionQueryZoom	Config-TC006 Zoom-TC003 Zoom-TC001, Zoom-TC002, Zoom-TC003, and Zoom-TC004 Zoom-TC002
Focus Near	Provide units with control receivers for DSP function of focus near	2.B.6	3.5.4 positionFocusLens 3.5.9 positionQueryFocus	Focus-TC001, Focus-TC002, Focus-TC003, and Focus-TC004 Focus-TC002
Focus Far	Provide units with control receivers for DSP function of focus far	2.B.6	3.5.4 positionFocusLens 3.5.9 positionQueryFocus	Focus-TC001, Focus-TC002, Focus-TC003, and Focus-TC004 Focus-TC002

Table 3. Requirements Traceability Matrix with Test Procedures (continued).

SS 6025 Closed Circuit Television Requirements Traceability Matrix				
User Need	**Requirement**	**TxDOT SS 6025 Reference**	**NTCIP Object Support**	**Test Procedure Identifier**
Manual and Auto Iris Control	Provide units with control receivers for DSP function of manual and auto iris control	2.B.6	3.6.4 systemLensFeatureControl 3.6.5 systemLensFeatureStatus 3.6.6 systemLensEquipped	Lens-TC002 and Lens-TC003 Lens-TC002 and Lens-TC003 Lens-TC001, Lens-TC002, and Lens-TC003
Iris Open	Provide units with control receivers for DSP function of iris open	2.B.6	3.2.10 rangeIrisLimit 3.3.5 timeoutIris 3.5.5 positionIrisLens 3.5.10 positionQueryIris	Config-TC008 Iris-TC003 Iris-TC001, Iris-TC002, Iris-TC003, and Iris-TC004 Iris-TC002
Iris Close	Provide units with control receivers for DSP function of iris close	2.B.6	3.2.10 rangeIrisLimit 3.3.5 timeoutIris 3.5.5 positionIrisLens 3.5.10 positionQueryIris	Config-TC008 Iris-TC003 Iris-TC001 Iris-TC002
Camera Power (Latching)	Provide units with control receivers for DSP function of camera power (latching)	2.B.6	3.6.1 systemCameraFeatureControl 3.6.2 systemCameraFeatureStatus 3.6.3 systemCameraEquipped	Features-TC002 Features-TC002 Features-TC001
Remote White Balancing Control	Provide units with control receivers for DSP function of white balancing control	2.B.6	None	
Auto and Manual White Balance Control	Provide units with control receivers for DSP function of auto and manual white balance control	2.B.6	None	

Table 3. Requirements Traceability Matrix with Test Procedures (continued).

SS 6025 Closed Circuit Television Requirements Traceability Matrix				
User Need	**Requirement**	**TxDOT SS 6025 Reference**	**NTCIP Object Support**	**Test Procedure Identifier**
Auxiliary Output	Provide units with control receivers for DSP function of one auxiliary output	2.B.6	3.9 CCTV Discrete Output Objects	Discrete-TC002
Quantities, Limits, and Values				
Label Color = White with Black Outline	Use Digital Signal Processing (DSP) for built-in I.D. Generator, with white letters and black outline	B.1	3.11.2.5 labelColor	Label-TC001 and PRL-TC001
Tilt Angle = +40° to -90°	Provide a unit with vertical movement of +40° to -90°	B.4	3.2.6 rangeTiltUpLimit 3.2.7 rangeTiltDownLimit	Config-TC005 and PRL-TC001 Config-TC005 and PRL-TC001
Tilt Speed = 20° per sec	Tilt speed must be 20° per sec	B.4	3.5.2 positionTilt	Tilt-TC001, Tilt-TC002, Tilt-TC003, and PRL-TC001
Pan Angle = 360	Provide a unit with horizontal movement of 360° full, contiguous rotation movement	B.4	3.2.2 rangePanLeftLimit 3.2.3 rangePanRightLimit	Config-TC003 and PRL-TC001 Config-TC003 and PRL-TC001
Pan Speed = 100° per sec	The pan speed must be up to 100° per sec.	B.4	3.5.1 positionPan	Pan-TC001 and PRL-TC001
Pan/Tilt Position Presets = 1	Provide units with remote control Pan/Tilt Position preset	B.6	3.2.1 rangeMaximumPreset	Config-TC002 and PRL-TC001

Table 3. Requirements Traceability Matrix with Test Procedures (continued).

SS 6025 Closed Circuit Television Requirements Traceability Matrix				
User Need	**Requirement**	**TxDOT SS 6025 Reference**	**NTCIP Object Support**	**Test Procedure Identifier**
Zoom and Focus Presets = 1	Provide units with remote control of Zoom and Focus position preset	B.6	3.2.1 rangeMaximumPreset	Config-TC002 and PRL-TC001
Auxiliary Outputs = 1	Provide units with remote control of one auxiliary output	B.6	3.9 CCTV Discrete Output Objects	Discrete-TC002 and PRL-TC001
Labels	Provide the built-in I.D. Generator that inserts camera I.D.	B.7	3.11.1 labelMaximum	Label-TC001 and PRL-TC001
Non-identified needs but covered in NTCIP 1205-CCTV				
Not Identified	Provide a true north reference	None	3.2.5 rangeTrueNorthOffset	Config-TC004
Not Identified	Provide units with remote control of one auxiliary output	None	3.8 CCTV Discrete Output Objects	Discrete-TC002
Not Identified	Provide units with remote control of heater	None	3.6.1 systemCameraFeatureControl 3.6.2 systemCameraFeatureStatus 3.6.3 systemCameraEquipped	Feature-TC003
Not Identified	Provide units with remote control of wiper	None	3.6.1 systemCameraFeatureControl 3.6.2 systemCameraFeatureStatus 3.6.3 systemCameraEquipped	Feature-TC004
Not Identified	Provide units with remote control of washer	None	3.6.1 systemCameraFeatureControl 3.6.2 systemCameraFeatureStatus 3.6.3 systemCameraEquipped	Feature-TC005
Not Identified	Provide units with remote control of blower	None	3.6.1 systemCameraFeatureControl 3.6.2 systemCameraFeatureStatus 3.6.3 systemCameraEquipped	Feature-TC006
Not Identified	Provide units with remote menuing	None	3.12.1 menuActivate 3.12.2 menuControl	Menu-TC001 Menu-TC001

Table 3. Requirements Traceability Matrix with Test Procedures (continued).

SS 6025 Closed Circuit Television Requirements Traceability Matrix				
User Need	**Requirement**	**TxDOT SS 6025 Reference**	**NTCIP Object Support**	**Test Procedure Identifier**
Not Identified	Provide units with remote configuration, control, and labeling of zones	None	3.10.3 zoneCameraEquipped 3.10.1 zoneMaximum 3.10.2 zoneTable 3.4.1 presetGotoPosition 3.4.2 presetStorePosition 3.4.3 presetPositionQuery	Config-TC010 Zone-TC002 Zone-TC002 Zone-TC001 and Zone-TC003 Zone-TC001 Zone-TC001
Traceable to NTCIP 1201-GLO				
Configuration				
Not Identified	Provide input and display of manufacturer information	None	2.2.2 globalMaxModules	Config-TC001 and GlobalConfig-TC001
			2.2.3 globalModuleTable [2.2.3.1 moduleNumber 2.2.3.2 moduleDeviceNode 2.2.3.2 moduleMake 2.2.3.2 moduleModel 2.2.3.2 moduleVersion 2.2.3.2 moduleType]	Config-TC001 and GlobalConfig-TC001
Not Identified	Provide indication of configuration change	None	2.2.1 globalSetIDParmeter	GlobalConfig-TC002
Security				
Not Identified	Provide administrative access to all data and user password definitions	None	2.7.1 communityNameAdmin	Security-TC001
Not Identified	Provide user access passwords	None	2.7.2 maxCommunityNames	Security-TC002
			2.7.3 communityNameTable	Security-TC002

Table 3. Requirements Traceability Matrix with Test Procedures (continued).

SS 6025 Closed Circuit Television Requirements Traceability Matrix				
User Need	**Requirement**	**TxDOT SS 6025 Reference**	**NTCIP Object Support**	**Test Procedure Identifier**
Traceable to NTCIP 2301-STMF				
Standardized Communications Protocol	Provide communications signals, data exchange protocol and timing that is compatible with the communications equipment.	B.6	2.7.1 communityNameAdmin 2.7.2 maxCommunityNames 2.7.3 communityNameTable	Security-TC001 Security-TC002 Security-TC002 *Note:* Numerous SNMP related functionality test procedures would be applicable to CCTV.
Not Identified	Provide generic ID of equipment and manufacturer	None	A.5.2 System Conformance Group	SNMP-TC041
Not Identified	Provide information related to application-level protocol troubleshooting	None	A.5.3 SNMP Statistics Conformance Group	SNMP-TC011
Not Identified	Accept a data exchange of at least 484 bytes	None	A.5.4 SNMP Configuration Group	SNMP-TC005
Traceable to NTCIP 2201-T2				
Standardized Communications Protocol	Provide communications signals, data exchange protocol and timing that is compatible with non-networked environment	None	Basic functionality of NULL does not require object support	*Note:* There are several NULL protocol-related test procedures. However, these do not appear to be applicable to CCTV.
Not Identified	Provide for single application-level protocol	None	Basic functionality of NULL does not require object support	

Table 3. Requirements Traceability Matrix with Test Procedures (continued).

SS 6025 Closed Circuit Television Requirements Traceability Matrix				
User Need	**Requirement**	**TxDOT SS 6025 Reference**	**NTCIP Object Support**	**Test Procedure Identifier**
Traceable to NTCIP 2101-PMPP/RS232				
Standardized Communications Protocol	Provide communications signals, data exchange protocol, and timing that is compatible with RS-232 Serial Port	None	A.7.1 lapBAdminTable A.7.1 lapBOperTable	232-TC025 232-TC025 *Note:* Numerous PMPP related functionality test procedures would be applicable to CCTV.
Not Identified	Provide information related to subnetwork-level protocol troubleshooting	None	A.7.2 rs232AsyncPortTable	232-TC024
Programmable Address	Provide each unit with a unique programmable address	B.6	Basic functionality of PMPP does not require object support	232-TC002
Group Address	Provide each unit with programmable group addresses	None	A.7.3 HDLC Group Address Conformance Group	232-TC013 and 232-TC014
Asynchronous	Data must be sent asynchronously		A.7.2 rs232PortTable	232-TC024
9600 Baud	Use a minimum of 9600 Baud		A.7.2 rs232PortTable	232-TC024

CCTV Test Procedures

One objective of the research project is to assist TxDOT in developing procedures. Appendix C provides a set of test procedures for testing CCTV pan, tilt, and zoom (PTZ) controllers. The test procedures include a prequalification test case that looks at general object support and supported values and 57 test cases that look at specific functionality expressed by the objects.

The prequalification test case retrieves minimum project requirements and maximum values, checks for whether the device implements the required objects, and performs a sampling of the supported values. The minimum project requirements and maximum values relate to those specified in the special specification on NTCIP for CCTV Equipment (Appendix B). These correspond to such things as the number of labels and the limits of panning, tilting, and zooming. The test case, as written, simply retrieves the value from a device. It does not make any value judgment as to whether the object meets the minimum required value. Although the test procedure could check for a specific minimum value, that interpretation is currently the responsibility of the person reviewing the test results. The object presence or instantiation test uses the information in a local MIB to indicate what objects should be present and then performs a read or SNMP "get" of all possible object instances. Whereas most test procedures use an SNMP "getNext" command to get the next object instantiated or "walk the MIB," the test case uses an SNMP "get" of specific instances. A management application would normally use an SNMP "get" to access information and the test case emulates that operation. The sampling test portion of the test uses an externally defined file of test values, and the test steps use a write or SNMP "set" to store the test value in a device.

The sampling test portion of the test case uses the external file of test values to customize the procedure to check compliance to TxDOT specifications rather than conformance to the NTCIP standard. From a NTCIP perspective, a device may limit the number of instances of an object. For example, the tilt up limit in NTCIP 1205-CCTV is 0 to 360 degrees. A manufacturer can subrange that limit to 10 degrees and still be compliant to the NTCIP standard. The requirement in SS 6025 is 40 degrees (*1*). To test conformance to a TxDOT specification, the test procedure should use a test value of 40. Basing the test case on externally defined values also allows districts to customize the test case to project-specific requirements.

The functional test cases use the Enterprise CCTV Test Procedures as their model (*15*). They test whether a device performs the function to which an NTCIP object maps. The organization of the test cases is as follows:

- Alarms
- Camera Configuration (Camera)
- Discrete Input and Output
- Camera Features
- Focus
- Global Configuration
- Iris
- Label
- Lens
- Menu
- Pan
- Security
- Tilt
- Zoom

Some of the functional areas and test cases go beyond the requirements of SS 6025 (*1*). The NTCIP standard addresses additional features that do not have a reference in the special specification. The test cases in Appendix C address some the additional features because some district and one-time use special specifications do reference those features.

The Global Configuration and Security test cases are not specific to CCTV. They would apply to any type of field device that claims compliance to NTCIP. The discussion of the communications and other test procedures appears in a separate section, and a listing appears in Appendix E.

CCTV Test Results

One of the framework recommendations that appears in the first year's report on this project is to use an NTCIP Profile Requirements List as a format for reporting test results. Appendix D is a filled out PRL showing the results of the Prequalification Test Case - TC001 in Appendix C. The PRL from NTCIP 1205:2001 v01.08 Amendment 1 v10 serves as the basic template, but with additional text, that clarifies some revision issues (*6*).

One of the purposes for including the PRL in NTCIP standards is to use it for selecting appropriate tests to check conformance. A manufacturer and user can also use it as a detailed indication of the capabilities of the implementation. NTCIP PRLs come with specific copyright permission to use them for creating a Protocol Implementation Conformance Specification (PICS). When a PRL includes specific object support and supported values information about an implementation, it becomes a PICS. As the name implies, it is a specification of the features in an implementation. Figure 1 illustrates how the test script changes the object support entries or adds supported values entries to the PRL to produce a PICS test report. Appendix D contains a full PICS test report using the PRL from NTCIP 1205-CCTV (6).

CCTV Configuration CONFORMANCE GROUP						
NTCIP 1205 Amend 1 Clause	**Object Name**	**Object Type**	**Object Status**	**Object Support**	**Allowed Values**	**Supported Values**
3.2, 3.3 and 3.11	**CCTV Configuration Conformance Group**	---	M	YES	----	-----
3.2	**CCTV Range Objects**	---	---	---	---	---
3.2.1	rangeMaximumPreset	S	3.2 : M	YES	0-255	64
3.2.2	rangePanLeftLimit	S	3.2 : M	YES	0-35999 | 65535	35999
3.2.3	rangePanRightLimit	S	3.2 : M	YES	0-35999 | 65535	35999
3.2.4	rangePanHomePosition	S	3.2 : M	YES	0-35999 | 65535	0
3.2.5	rangeTrueNorthOffset	P	3.2 : M	YES	0-35999 | 65535	PASSED: 0,1,18000, 35998,359 99,36000, FAILED: -1,
3.2.6	rangeTiltUpLimit	S	3.2 : M	YES	0-35999 | 65535	1500
3.2.7	rangeTiltDownLimit	S	3.2 : M	YES	0-35999 | 65535	27000
3.2.8	rangeZoomLimit	S	3.2 : M	YES	0-65535	65535

Figure 1. CCTV PRL Information Test Case Results.

The results of the other test procedures and test cases listed in Appendix C would follow the same principle of reusing existing documentation. Figure 2 shows how test case results of the Cabinet Alarm test case would appear as a test result report. Reporting results in this manner not only indicates what was tested and whether it passed or not but also documents the specifics of how it was tested. (A future revision to the test script will add the test script revision number to the report as well.)

<table>
<tr><td>Test Case:
Alarm-TC001</td><td colspan="4">Title: Cabinet Alarm
Description: This test case tests the cabinet open alarm and the label associated with it.
Variables:
Pass/Fail Criteria: The device under test (DUT) shall pass every verification step included within the Test Case in order to pass the Test Case.</td></tr>
<tr><td>Test Step Number</td><td colspan="3">Test Procedure</td><td>Results</td></tr>
<tr><td>1</td><td colspan="3">SET
labelText.<alarmCLabIndex> = <alarmCLabText1>
labelHeight.<alarmCLabIndex> = <alarmCLabHeight1>
labelColor.<alarmCLabIndex> = <alarmCLabColor1>
labelStartRow.<alarmCLabIndex> = <alarmCLabStartRow1>
labelStartColumn.<alarmCLabIndex> = <alarmCLabStartColumn1></td><td>Pass</td></tr>
<tr><td>2</td><td colspan="3">SET alarmLabelIndex.0 to <alarmCLabIndex> 00 00 00 00 00 00</td><td>Pass</td></tr>
<tr><td>3</td><td colspan="3">USER VERIFY that no labels are being shown.</td><td>Pass</td></tr>
<tr><td>4</td><td colspan="3">SET alarmLatchClear.0 to 0x00</td><td>Pass</td></tr>
<tr><td>5</td><td colspan="3">Turn on the alarm and USER VERIFY the label for the alarm is shown.</td><td>Pass</td></tr>
<tr><td>6</td><td colspan="3">GET alarmStatus.0 and alarmLatchStatus.0</td><td>Pass</td></tr>
<tr><td>7</td><td colspan="3">VERIFY RESPONSE VALUE
alarmStatus = 0x80
alarmLatchStatus = 0x80</td><td>Pass</td></tr>
<tr><td>8</td><td colspan="3">SET alarmLatchClear.0 to 0x00</td><td>Pass</td></tr>
<tr><td>9</td><td colspan="3">GET alarmStatus.0 and alarmLatchStatus.0</td><td>Pass</td></tr>
<tr><td>10</td><td colspan="3">VERIFY RESPONSE VALUE
alarmStatus = 0x80
alarmLatchStatus = 0x80</td><td>Pass</td></tr>
<tr><td>11</td><td colspan="3">USER VERIFY the label for the alarm is shown and deactivate the alarm.</td><td>Pass</td></tr>
<tr><td>12</td><td colspan="3">USER VERIFY the label for the alarm is off.</td><td>Pass</td></tr>
<tr><td>13</td><td colspan="3">GET alarmStatus.0 and alarmLatchStatus.0</td><td>Pass</td></tr>
<tr><td>14</td><td colspan="3">VERIFY Response Value
alarmStatus = 0x00
alarmLatchStatus = 0x80</td><td>Pass</td></tr>
<tr><td>15</td><td colspan="3">USER VERIFY the label for the alarm is off.</td><td>Pass</td></tr>
<tr><td>16</td><td colspan="3">SET alarmLatchClear.0 to 0x00</td><td>Pass</td></tr>
<tr><td>17</td><td colspan="3">GET alarmStatus.0 and alarmLatchStatus.0</td><td>Pass</td></tr>
<tr><td>18</td><td colspan="3">VERIFY RESPONSE VALUE
alarmStatus = 0x00
alarmLatchStatus = 0x00</td><td>Pass</td></tr>
<tr><td colspan="5">Test Case Results</td></tr>
<tr><td>Tested By:</td><td>Jeremy Johnson</td><td>Date Tested:</td><td>4/22/06</td><td>Pass</td></tr>
<tr><td>Test Case Notes:</td><td colspan="4">Cohu I-Dome Camera with I-Control SSN 449497</td></tr>
<tr><td>Version History:</td><td colspan="4">V1.0 – Initial Draft 09/20/05
V1.1 – Removed deprecated labelFontType 11/03/05</td></tr>
</table>

Figure 2. CCTV Cabinet Alarm Test Case Results Form.

COMMUNICATIONS LEVEL TEST PROCEDURES

In order to communicate, any specification of an ITS field device needs to address requirements for the communications protocols and physical interfaces. For compliance to NTCIP, this essentially consists of specifying one or more or the application, transport, and subnetwork-level standards and the plant level physical interface. The NTCIP Guide provides information on how to select the various standards from each level and the options to use (*12*).

All of the communications level standards include functional requirements. Some of them also include object conformance groups. If these functional requirements and conformance groups appear in a TxDOT specification, then there are test procedures that would apply. A number of test procedures that relate to communications are already in the public domain. The following sections provide general descriptions of what is available. Appendix E provides further details on individual test cases.

Application Level

There are a number of test procedures that validate the proper operation of the SNMP protocol that apply to an implementation conformant to NTCIP 2301-STMF (*8*). The functional areas that they address are:

- General SNMP Commands
- Error Responses to Commands
- Community Name Validation
- Statistical Information
- Standard Data Type Encoding
- Opaque Ending

An implementation may incorporate the STMP protocol that NTCIP 2301-STMF specifies (*8*). A number of test procedures are applicable if that is the case. The test procedures address the Encoding and Decoding of the Data Types

Transport Level

Several test procedures are available for the NTCIP 2201 – Transportation Transport Profile (NTCIP 2201-T2) (9). These procedures address:

- Invalid Protocol Identifier
- Maximum Packet Size
- Support for the ipNetToMedia Conformance Group

Subnetwork Level

At the subnetwork-level, procedures for checking the functionality and objects in two conformance groups may be applicable. The test procedures for PMPP as defined in NTCIP 2101-PMPP/RS232 are:

- Short Address Validation
- Long Address Support and Validation
- Broadcast and Polling
- Group Address Support and Validation
- Polling
- Control Byte
- Invalid Protocol Identifier
- Field Check Sum
- RS232 and HDLC Conformance Groups
- Frame Size and Buffering
- Data Rates and Response Time

TRAFFIC SIGNAL CONTROLLER DOCUMENTATION

TxDOT Specifications for Traffic Signal Controllers

TxDOT DMS 11170-TSC defines the general and the NTCIP specific requirements for traffic signal controllers and their assemblies (2). It covers both the physical, environmental, and functional requirements and the NTCIP requirements. The document devotes about one-tenth of its material to NTCIP compliance. It follows the NTCIP Guide's recommendations on how to spell out requirements. Besides referencing NTCIP 1202 – Object Definitions for Actuated Signal Controller Units (NTCIP 1202-ASC), DMS 11170-TSC also includes an Object Range Value table as suggested in the NEMA TS 2 Standard (*16*,*2*,*17*).

There are three suggestions for additions to DMS 11170-TSC:

- Reference communications-level conformance groups and object definitions.
- Refer to NTCIP 2103-PPP for dial-up communications (*13*).
- Refer to NTCIP 2104-Ethernet for Ethernet communications (*18*).

Several of the communications-level standards contain conformance groups and object definitions that relate to parameters and status information at that level. The standards themselves designate certain groups as being mandatory, but some implementations do not include them. For example, there are object conformance groups related to system information, SNMP statistics, and HDLC that may apply. Sections 2.F and 2.G of Appendix B provide example wording of what to include. Please also keep in mind that different transport and subnetwork-level protocols have different conformance groups.

DMS 11170-TSC references dial-up communications but does not cite any particular standard to follow (*2*). NTCIP 2103-PPP defines the protocol for use in this type of point-to-point communication (*13*). It also defines an authentication protocol that protects against unauthorized access. The newest version of NTCIP 2103-PPP includes object definitions that standardize the information related to modem initialization and phone numbers (*13*).

Although it is not required by any NTCIP standard, most current state-of-the-art traffic signal controllers include an Ethernet interface. A number of recent projects involving DMSs and CCTVs include a reference to Ethernet (*19*,*20*). While there may not be an immediate need to address an Ethernet interface, DMS 11170-TSC could list it as optional (*2*).

Initial Requirements Traceability Matrix

Developing a full requirements traceability matrix for a traffic signal controller is beyond the scope of this project. This is especially true if the RTM needs to include user needs and provide cross references to a full set of test case identifiers. However, DMS 11170-TSC does have a set of NTCIP requirements (*2*). This research prototyped some diamond-controller test procedures, and other applicable procedures exist. Given these limitations and available information, the researcher does provide an initial RTM. A first-order RTM also provides some insight into the work effort needed to develop a full set of test procedures.

In Table 4, an RTM enumerates the NTCIP requirements in DMS 11170-TSC, cites the clause or heading defining the requirement, and lists the NTCIP objects that relate to the requirement. In most cases, the test procedure identifier is blank. However, where one of the test cases in Appendix G is applicable, the matrix cross-references it.

Several of the entries in the Test Procedure Identifier column add a prefix to some of the test case identifiers. The prefix is a reference to the device type source. For example, the globalMaxModules object lists (CCTV) Config-TC001 and (ASC) GloCon-TC001, which are equivalent and equally applicable. The column entry lists both because, currently each group or organization working on test procedures for a specific ITS field device is generating its own set of procedures for the objects that appear in NTCIP 1201-GLO (*7*). The NTCIP working groups are trying to resolve this duplication. However, TxDOT will have a similar situation. Is there a single set of procedures that can apply to all field devices that implement part of NTCIP 1201-GLO (*7*)? What about procedures for SNMP, RS-232, and the other communications level standards that can apply to a number of ITS field devices? The researcher believes that there should be a single set of procedures that test for compliance and/or conformance of the common conformance groups and protocols. However, until there is research into whether this is possible, each ITS field device may end up with its own set of test procedures.

Table 4. TSC Requirements Traceability Matrix with Test Procedures.

DMS 11170-TSC Requirements Traceability Matrix					
User Need	**Requirement**	**TxDOT DMS 11170 Reference**	**NTCIP Object Support**		**Test Procedure Identifier** [1]
Traceable to NTCIP 1202-ASC					
Phase Conformance Group					
	Implement all mandatory objects and all mandatory conformance groups defined in "Actuated Signal Controller Object Definitions," NTCIP 1202:1996: Phase Conformance Group	NTCIP Compliance	maxPhases		PRL-TC001[2]
			phaseTable		PRL-TC001
				phaseNumber	DetOps-TC001 – DetOps-TC018[3]
				phaseWalk	
				phasePedestrianClear	
				phaseMinimumGreen	DetOps-TC001 – DetOps-TC018
				phasePassage	DetOps-TC001 – DetOps-TC018
				phaseMaximum1	DetOps-TC001 – DetOps-TC018
				phaseMaximum2	
				phaseYellowChange	DetOps-TC001 – DetOps-TC018
				phaseRedClear	DetOps-TC001 – DetOps-TC018

[1] Unless otherwise stated, the Test Procedure Identifier refers to those listed in Appendix G.
[2] The Protocol/Profile Requirements test case also checks for support of all objects listed in the MIB.
[3] The Detector Operations test cases do not test the phase intervals but rely on specific values.

Table 4. TSC Requirements Traceability Matrix with Test Procedures (continued).

DMS 11170-TSC Requirements Traceability Matrix					
User Need	**Requirement**	**TxDOT DMS 11170 Reference**	**NTCIP Object Support**		**Test Procedure Identifier**
	Not Required[4]			phaseRedRevert	
	Implement all mandatory objects and all mandatory conformance groups defined in "Actuated Signal Controller Object Definitions," NTCIP 1202:1996: Phase Conformance Group			phaseAddedInitial	
				phaseMaximumInitial	
				phaseTimeBeforeReduction	
	Not Required[5]			phaseCarsBeforeReduction	
	Implement all mandatory objects and all mandatory conformance groups defined in "Actuated Signal Controller Object Definitions," NTCIP 1202:1996: Phase Conformance Group			phaseTimeToReduce	
	Not Required [5]			phaseReduceBy	
	Implement all mandatory objects and all mandatory conformance groups defined in "Actuated Signal Controller Object Definitions," NTCIP 1202:1996: Phase Conformance Group			phaseMinimumGap	
	Implement the following optional			phaseDynamicMaxLimit	

[4] Optional object not required by DMS 11170-TSC.
[5] Optional object not required by DMS 11170-TSC.

Table 4. TSC Requirements Traceability Matrix with Test Procedures (continued).

DMS 11170-TSC Requirements Traceability Matrix					
User Need	**Requirement**	**TxDOT DMS 11170 Reference**	**NTCIP Object Support**		**Test Procedure Identifier**
	objects as defined in the "Actuated Signal Controller Object Definitions," NTCIP 1202:1996: phaseDynamicMaxLimit and phase DynamicMaxStep.			phaseDynamicMaxStep	
	Implement all mandatory objects and all mandatory conformance groups defined in "Actuated Signal Controller Object Definitions," NTCIP 1202:1996: Phase Conformance Group			phaseStartup	DCT-T0001[6]
				phaseOptions	
				phaseRing	
				phaseConcurrency	
			maxPhaseGroups		PRL-TC001
			phaseStatusGroup Table		PRL-TC001
				phaseStatusGroupNumber	
				phaseStatusGroupReds	IM-TC0001
				phaseStatusGroupYellows	IM-TC0001
				phaseStatusGroupGreens	DetOps-TC001 – DetOps-TC018, IM-TC0001
				phaseStatusGroupDont Walks	IM-TC0001
				phaseStatusGroupPedClears	IM-TC0001
				phaseStatusGroupWalks	IM-TC0001
				phaseStatusGroupVehCalls	IM-TC0001
				phaseStatusGroupPedCalls	IM-TC0001

[6] The dbCreateTransaction test case only looks at the phaseStartup intervals.

Table 4. TSC Requirements Traceability Matrix with Test Procedures (continued).

DMS 11170-TSC Requirements Traceability Matrix					
User Need	**Requirement**	**TxDOT DMS 11170 Reference**	**NTCIP Object Support**		**Test Procedure Identifier**
				phaseStatusGroupPhaseOns	Seg-TC001 DetOps-TC001 – DetOps-TC018, IM-TC0001
				phaseStatusGroup PhaseNexts	IM-TC0001
			phaseStatus ControlTable		PRL-TC001
				phaseControlGroupNumber	
				phaseControlGroup PhaseOmit	
				phaseControlGroupPedOmit	
				phaseControlGroupHold	
				phaseControlGroup ForceOff	
				phaseControlGroupVehCall	DetOps-TC001 – DetOps-TC018
				phaseControlGroupPedCall	
Detector					
	Implement all mandatory objects and all mandatory conformance groups defined in “Actuated Signal Controller Object Definitions,” NTCIP 1202:1996: Detector Conformance Group	NTCIP Compliance	maxVehicleDetectors		PRL-TC001
			vehicleDetectorTable		PRL-TC001
				vehicleDetectorNumber	
				vehicleDetectorOptions	
				vehicleDetectorCallPhase	
				vehicleDetectorSwitchPhase	
				vehicleDetectorDelay	

Table 4. TSC Requirements Traceability Matrix with Test Procedures (continued).

DMS 11170-TSC Requirements Traceability Matrix					
User Need	**Requirement**	**TxDOT DMS 11170 Reference**	**NTCIP Object Support**		**Test Procedure Identifier**
				vehicleDetectorExtend	DetOps-TC001 – DetOps-TC018
	Implement the following optional objects as defined in the "Actuated Signal Controller Object Definitions," NTCIP 1202:1996			vehicleDetectorQueueLimit	
	Implement all mandatory objects and all mandatory conformance groups defined in "Actuated Signal Controller Object Definitions," NTCIP 1202:1996: Detector Conformance Group			vehicleDetectorNoActivity	
				vehicleDetector MaxPresence	
				vehicleDetector ErraticCounts	
	Implement the following optional objects as defined in the "Actuated Signal Controller Object Definitions," NTCIP 1202:1996			vehicleDetectorFailTime	
	Implement all mandatory objects and all mandatory conformance groups defined in "Actuated Signal Controller Object Definitions," NTCIP 1202:1996: Detector Conformance Group			vehicleDetectorAlarms	
	Implement the following optional objects as defined in the "Actuated Signal Controller Object Definitions," NTCIP 1202:1996			vehicleDetectorReported Alarms	

Table 4. TSC Requirements Traceability Matrix with Test Procedures (continued).

DMS 11170-TSC Requirements Traceability Matrix					
User Need	**Requirement**	**TxDOT DMS 11170 Reference**	**NTCIP Object Support**		**Test Procedure Identifier**
	Definitions," NTCIP 1202:1996				
	Implement all mandatory objects and all mandatory conformance groups defined in "Actuated Signal Controller Object Definitions," NTCIP 1202:1996: Detector Conformance Group			vehicleDetectorReset	
			maxVehicle Detector StatusGroups		PRL-TC001
			vehicleDetector StatusGroupTable		PRL-TC001
				vehicleDetectorStatusGroup Number	
				vehicleDetectorStatusGroup Active	
				vehicleDetectorStatusGroup Alarms	
			maxPedestrian Detectors		PRL-TC001
			pedestrian DetectorTable		PRL-TC001
				pedestrianDetectorNumber	
				pedestrianDetector CallPhase	
				pedestrianDetector NoActivity	
				pedestrianDetector MaxPresence	
				pedestrianDetector ErraticCounts	
				pedestrianDetector Alarms	

Table 4. TSC Requirements Traceability Matrix with Test Procedures (continued).

DMS 11170-TSC Requirements Traceability Matrix					
User Need	**Requirement**	**TxDOT DMS 11170 Reference**	**NTCIP Object Support**		**Test Procedure Identifier**
Volume Occupancy Report					
	Implement all mandatory objects of all optional conformance groups as defined in NTCIP 1202:1996: Volume Occupancy Report Conformance Group	NTCIP Compliance	volume OccupancySequence		
			volume OccupancyPeriod		
			activeVolume OccupancyDetectors		
			Volume OccupancyTable		PRL-TC001
				detectorVolume	
				detectorOccupancy	
Unit					
	Implement all mandatory objects of all optional conformance groups as defined in NTCIP 1202:1996: Unit Conformance Group	NTCIP Compliance	unitStartUpFlash		
			unitAutoPedestrian Clear		
			unitBackupTime		
	Implement the following optional objects as defined in the “Actuated Signal Controller Object Definitions,” NTCIP 1202:1996		unitRedRevert		
	Implement all mandatory objects of all optional conformance groups as defined in NTCIP 1202:1996: Unit Conformance Group		unitControlStatus		IM-TC0001
			unitFlashStatus		
			unitAlarmStatus2		
			unitAlarmStatus1		

Table 4. TSC Requirements Traceability Matrix with Test Procedures (continued).

DMS 11170-TSC Requirements Traceability Matrix					
User Need	**Requirement**	**TxDOT DMS 11170 Reference**	**NTCIP Object Support**		**Test Procedure Identifier**
			shortAlarmStatus		IM-TC0001
			unitControl		
	Implement the following optional objects as defined in the “Actuated Signal Controller Object Definitions,” NTCIP 1202:1996		maxAlarmGroups		
			alarmGroupTable		
				alarmGroupNumber	
				alarmGroupState	
Special Function					
	Implement all mandatory objects of all optional conformance groups as defined in NTCIP 1202:1996: Special Function Conformance Group	NTCIP Compliance	maxSpecialFunction Outputs		PRL-TC001
			specialFunctionOutput Table		PRL-TC001
				specialFunctionOutput Number	
				specialFunctionOutput Control	
				specialFunctionOutput Status	
Coordination					
	Implement all mandatory objects of all optional conformance groups as defined in NTCIP 1202:1996: Coordination Conformance Group	NTCIP Compliance	coordOperational Mode		
			coordCorrectionMode		
			coordMaximumMode		
			coordForceMode		
			maxPatterns		
			patternTableType		
			patternTable		
				patternNumber	

Table 4. TSC Requirements Traceability Matrix with Test Procedures (continued).

DMS 11170-TSC Requirements Traceability Matrix					
User Need	**Requirement**	**TxDOT DMS 11170 Reference**	**NTCIP Object Support**		**Test Procedure Identifier**
				patternCycleTime	
			patternOffsetTime		
			patternSplitNumber		
			patternSequence Number		
			maxSplits		PRL-TC001
			splitTable		PRL-TC001
				splitNumber	
				splitPhase	
				splitTime	
				splitMode	
				splitCoordPhase	
			coordPatternStatus		IM-TC0001
			localFreeStatus		
			coordCycleStatus		
			coordSyncStatus		
			systemPatternControl		
			systemSyncControl		
Time Base					
	Implement all mandatory objects of all optional conformance groups as defined in NTCIP 1202:1996: Time Base Conformance Group	NTCIP Compliance	timebaseAscPattern Sync		
			maxTimebaseAsc Actions		PRL-TC001
			timebaseAscAction Table		PRL-TC001
				timebaseAscActionNumber	

Table 4. TSC Requirements Traceability Matrix with Test Procedures (continued).

DMS 11170-TSC Requirements Traceability Matrix					
User Need	**Requirement**	**TxDOT DMS 11170 Reference**	**NTCIP Object Support**		**Test Procedure Identifier**
				timebaseAscPattern	
				timebaseAscAuxillary Function	
				timebaseAscSpecial Function	
				timebaseAscActionStatus	
Preempt					
	Implement all mandatory objects of all optional conformance groups as defined in NTCIP 1202:1996: Preempt Conformance Group	NTCIP Compliance	maxPreempts		PRL-TC001
			PreemptTable		PRL-TC001
				preemptNumber	
				preemptControl	
				preemptLink	
				preemptDelay	
				preemptMinimumDuration	
	Implement the following optional objects as defined in the "Actuated Signal Controller Object Definitions," NTCIP 1202:1996			preemptMinimumGreen	
				preemptMinimumWalk	
				preemptEnterPedClear	
	Implement all mandatory objects of all optional conformance groups as defined in NTCIP 1202:1996: Preempt Conformance Group			preemptTrackGreen	
				preemptDwellGreen	
				preemptMaximumPresence	
				preemptTrackPhase	
				preemptDwellPhase	
	Not Required[7]	N/A		preemptDwellPed	

[7] Optional object not required by DMS 11170-TSC

Table 4. TSC Requirements Traceability Matrix with Test Procedures (continued).

DMS 11170-TSC Requirements Traceability Matrix					
User Need	**Requirement**	**TxDOT DMS 11170 Reference**	**NTCIP Object Support**		**Test Procedure Identifier**
	Implement all mandatory objects of all optional conformance groups as defined in NTCIP 1202:1996: Preempt Conformance Group	NTCIP Compliance		preemptExitPhase	
	Implement the following optional objects as defined in the "Actuated Signal Controller Object Definitions," NTCIP 1202:1996			preemptState	
	Not Required[8]	N/A		preemptTrackOverlap	
				preemptDwellOverlap	
				preemptCyclingPhase	
				preemptCyclingPed	
				preemptCyclingOverlap	
				preemptEnterYellowChange	
				preemptEnterRedClear	
				preemptTrackYellow Change	
				preemptTrackRedClear	
	Implement the following optional objects as defined in the "Actuated Signal Controller Object Definitions," NTCIP 1202:1996	NTCIP Compliance	preemptControlTable		PRL-TC001
				preemptControlNumber	
				preemptControlState	
Ring					

[8] These objects were added in NTCIP 1202 v02.18.

Table 4. TSC Requirements Traceability Matrix with Test Procedures (continued).

DMS 11170-TSC Requirements Traceability Matrix					
User Need	**Requirement**	**TxDOT DMS 11170 Reference**	**NTCIP Object Support**		**Test Procedure Identifier**
	Implement all mandatory objects of all optional conformance groups as defined in NTCIP 1202:1996: Ring Conformance Group	NTCIP Compliance	maxRings		PRL-TC001
			maxSequences		PRL-TC001
			sequenceTable		PRL-TC001
				sequenceNumber	
				sequenceRingNumber	
				sequenceData	
			maxRingControl Groups		PRL-TC001
			ringControlGroup Table		PRL-TC001
				ringControlGroupNumber	
				ringControlGroupStopTime	
	Implement the following optional objects as defined in the “Actuated Signal Controller Object Definitions” NTCIP 1202:1996			ringControlGroupForceOff	
				ringControlGroupMax2	
				ringControlGroup MaxInhibit	
	Implement all mandatory objects of all optional conformance groups as defined in NTCIP 1202:1996: Ring Conformance Group			ringControlGroupPed Recycle	
	Not Required[9]			ringControlGroupRedRest	
				ringControlGroupOmitRed Clear	
	Not Required[10]	N/A	ringStatusTable		PRL-TC001

[9] These optional objects are not required by DMS 11170-TSC.
[10] This table and object were added in NTCIP 1202 v02.18.

Table 4. TSC Requirements Traceability Matrix with Test Procedures (continued).

DMS 11170-TSC Requirements Traceability Matrix					
User Need	**Requirement**	**TxDOT DMS 11170 Reference**	**NTCIP Object Support**		**Test Procedure Identifier**
				ringStatus	Seg-TC001 DetOps-TC001 – DetOps-TC018, IM-TC0001
Channel					
	Implement all mandatory objects of all optional conformance groups as defined in NTCIP 1202:1996: Channel Conformance Group	NTCIP Compliance	maxChannels		
			channelTable		PRL-TC001
				channelNumber	
				channelControlSource	
				channelControlType	
				channelFlash	
				channelDim	
			maxChannelStatusGroups		PRL-TC001
			channelStatusGroupTable		PRL-TC001
				channelStatusGroupNumber	
				channelStatusGroupReds	
				channelStatusGroupYellows	
				channelStatusGroupGreens	
Overlap					
	Implement all mandatory objects of all optional conformance groups as defined in NTCIP 1202:1996: Overlap Conformance Group	NTCIP Compliance	maxOverlaps		PRL-TC001
			overlapTable		PRL-TC001
				overlapNumber	
				overlapType	
				overlapIncludedPhases	
				overlapModifierPhases	

Table 4. TSC Requirements Traceability Matrix with Test Procedures (continued).

DMS 11170-TSC Requirements Traceability Matrix					
User Need	**Requirement**	**TxDOT DMS 11170 Reference**	**NTCIP Object Support**		**Test Procedure Identifier**
				overlapTrailGreen	
				overlapTrailYellow	
				overlapTrailRed	
			maxOverlapStatus Groups		PRL-TC001
			overlapStatusGroup Table		PRL-TC001
				overlapStatusGroupNumber	
				overlapStatusGroupReds	
				overlapStatusGroupYellows	
				overlapStatusGroupGreens	DetOps-TC001 – DetOps-TC018
TS 2 Port 1					
	Implement all mandatory objects of all optional conformance groups as defined in NTCIP 1202:1996: TS 2 Port 1 Conformance Group	NTCIP Compliance	maxPort1Addresses		PRL-TC001
			port1Table		PRL-TC001
				port1Number	
				port1DevicePresent	
				port1Frame40Enable	
				port1Status	
				port1FaultFrame	
Block Objects					
	Not Required[11]	N/A	ascBlockGetControl		
			ascBlockData		
			ascBlockErrorStatus		

[11] These objects were added in NTCIP 1202 v02.18.

Table 4. TSC Requirements Traceability Matrix with Test Procedures (continued).

DMS 11170-TSC Requirements Traceability Matrix					
User Need	**Requirement**	**TxDOT DMS 11170 Reference**	**NTCIP Object Support**		**Test Procedure Identifier**
Traceable to NTCIP 1201-GLO					
Time Management					
	Implement all mandatory objects of all optional conformance groups as defined in “Global Object Definitions,” NTCIP 1201:1996: Time Management Conformance Group	NTCIP Compliance	globalTme		TAD-TC0001
			globalDayLight Savings		TAD-TC0001
	Deprecated objects[12]		globalLocalTimeDifferential		TAD-TC0001
Time Base Event Schedule					
	Implement all mandatory objects of all optional conformance groups as defined in “Global Object Definitions,” NTCIP 1201:1996: Timebase Event Schedule Conformance Group		maxTimeBase ScheduleEntries		PRL-TC001
			timeBaseSchedule Table		PRL-TC001
				timeBaseSchedule Number	
				timeBaseScheduleMonth	
				timeBaseScheduleDay	
				timeBaseScheduleDate	
				timeBaseScheduleDayPlan	
	Not Required[13]	N/A	timeBaseSchedule Table-status		

[12] This object was deprecated in NTCIP 1201 v02.26.
[13] This object was added in NTCIP 1201 v02.26.

Table 4. TSC Requirements Traceability Matrix with Test Procedures (continued).

DMS 11170-TSC Requirements Traceability Matrix					
User Need	**Requirement**	**TxDOT DMS 11170 Reference**	**NTCIP Object Support**		**Test Procedure Identifier**
	Implement all mandatory objects of all optional conformance groups as defined in "Global Object Definitions," NTCIP 1201:1996: Timebase Event Schedule Conformance Group	NTCIP Compliance	maxDayPlans		PRL-TC001
			maxDayPlanEvents		PRL-TC001
			timeBaseDayPlan Table		PRL-TC001
				dayPlanEventNumber	
				dayPlanHour	
				dayPlanMinute	
				dayPlanActionNumber OID	
			dayPlanStatus		
	Not Required[14]	N/A	controller-standardTimeZone		TAD-TC0001
	Not Required[15]		controller-localTime		
Database Management and Configuration Management					
	Implement all mandatory objects of all optional conformance groups as defined in "Global Object Definitions," NTCIP 1201:1996: Database Management Conformance Group	NTCIP Compliance	dbCreateTransaction		DCT-TC0001
	Deprecated objects[16]		dbErrorType		
			dbErrorID		
			dbTransactionID		

[14] This object was added in NTCIP 1201 v02.26.
[15] This object was added in NTCIP 1201 v02.26.
[16] These objects were deprecated in NTCIP 1201:1996 v01.10.

Table 4. TSC Requirements Traceability Matrix with Test Procedures (continued).

DMS 11170-TSC Requirements Traceability Matrix				
User Need	**Requirement**	**TxDOT DMS 11170 Reference**	**NTCIP Object Support**	**Test Procedure Identifier**
	Implement the following optional objects as defined in the “Global Object Definitions,” NTCIP 1201:1996 [17]		dbMakeID	
	Implement all mandatory objects of all optional conformance groups as defined in “Global Object Definitions,” NTCIP 1201:1996: Database Management Conformance Group		dbVerifyStatus	DCT-TC0001
			dbVerifyError	
Configuration (Global)				
	Implement all mandatory objects of all mandatory conformance groups as defined in the “Global Object Definitions,” NTCIP 1201:1996: Global Configuration Conformance Group	NTCIP Compliance	globalSetIDParmeter	(ASC)GloCon-TC002
			globalMaxModules	(CCTV)Config-TC001, (ASC) GloCon-TC001 PRL-TC001
			globalModuleTable	PRL-TC001
			moduleNumber	(ASC) GloCon-TC001
			moduleDeviceNode	
			moduleMake	
			moduleModel	

[17] This object was deprecated in NTCIP 1201:1996 v01.10.

Table 4. TSC Requirements Traceability Matrix with Test Procedures (continued).

DMS 11170-TSC Requirements Traceability Matrix					
User Need	**Requirement**	**TxDOT DMS 11170 Reference**	**NTCIP Object Support**		**Test Procedure Identifier**
				moduleVersion	
				moduleType	
	Not Required[18]	N/A		controller-baseStandards	
Report					
	Implement all mandatory objects of all optional conformance groups as defined in the “Global Object Definitions,” NTCIP 1201:1996: Report Conformance Group	NTCIP Compliance	maxEventLogConfigs		PRL-TC001
			eventLogConfigTable		PRL-TC001
				eventConfigID	RLD-TC001 – RLD-TC0008
				eventConfigClass	
				eventConfigMode	
				eventConfigCompareValue	
				eventConfigCompareValue2	
				eventConfigCompareOID	
	Implement the following optional objects as defined in the “Global Object Definitions,” NTCIP 1201:1996			eventConfigLogOID	
				eventConfigAction	
	Not Required[19]	N/A		eventConfigStatus	
	Implement all mandatory objects of all optional conformance groups as defined in the “Global Object Definitions,” NTCIP 1201:1996: Report Conformance Group	NTCIP Compliance	maxEventLogSize		PRL-TC001
			eventLogTable		PRL-TC001
				eventLogClass	RLD-TC001 – RLD-TC0008
				eventLogNumber	
				eventLogID	
				eventLogTime	
				eventLogValue	

[18] This object was added in NTCIP 1201 v02.26.
[19] This object was added in NTCIP 1201 v02.26.

Table 4. TSC Requirements Traceability Matrix with Test Procedures (continued).

DMS 11170-TSC Requirements Traceability Matrix					
User Need	**Requirement**	**TxDOT DMS 11170 Reference**	**NTCIP Object Support**		**Test Procedure Identifier**
			maxEventClasses		PRL-TC001
			eventClassTable		PRL-TC001
				eventClassNumber	RLD-TC001 – RLD-TC0008
				eventClassLimit	
				eventClassClearTime	
	Implement the following optional objects as defined in the “Global Object Definitions,” NTCIP 1201:1996			eventClassDescription	
	Implement all mandatory objects of all optional conformance groups as defined in the “Global Object Definitions,” NTCIP 1201:1996: Report Conformance Group			eventClassNumRowsInLog	
	Not Required[20]	N/A	numEvents		
STMP					
	Implement all mandatory objects of all optional conformance groups as defined in the “Global Object Definitions,” NTCIP 1201:1996: STMF (STMP) Conformance Group	NTCIP Compliance	dynamicObject Persistence		

[20] This object was added in NTCIP 1201 v02.26.

Table 4. TSC Requirements Traceability Matrix with Test Procedures (continued).

DMS 11170-TSC Requirements Traceability Matrix					
User Need	**Requirement**	**TxDOT DMS 11170 Reference**	**NTCIP Object Support**		**Test Procedure Identifier**
PMPP					
	Implement all mandatory objects of all optional conformance groups as defined in the “Global Object Definitions,” NTCIP 1201:1996: PMPP Conformance Group	NTCIP Compliance	maxGroupAddresses		PRL-TC001
			hdlcGroupAddress Table	hdlcGroupAddressIndex	232-TC013 and 232-TC014
				hdlcGroupAddress	
Security					
	Implement all mandatory objects of all mandatory conformance groups as defined in the “Global Object Definitions,” NTCIP 1201:1996: Security Conformance Group	None	communityName Admin		(ASC) Security-TC001
			communityNamesMax		(ASC) Security-TC002
			communityNameTable	communityNameIndex	(ASC) Security-TC002
				communityNameUser	
				communityNameAccess Mask	

Test Plan and Documentation

Since a task of developing NTCIP test procedures for one of the ITS field devices can be a significant project in and of itself, the researcher found that following some of the recommendations in the Institute of Electrical and Electronics Engineers (IEEE) Std. 829 – IEEE Standard for Software Test Documentation are useful (*3*). Prior to actually writing procedures, the IEEE standard suggests the development of an overall plan, one or more test design specifications, and test case specifications. The overall plan conveys the scope, approach, resources, and schedule of testing activities. Its primary purpose is to present a high-level view of the project to inform all interested parties. The test design specifications provide a more detailed view of the testing project. A test engineer's supervisor and any group such as a project monitoring committee uses the test design specifications to make sure that a test engineer understands the projects and is addressing what is needed. This serves as part of the validation step in the project development. Test case specifications then outline individual test cases that verify specific features and functions of an implementation undergoing testing. The test case specifications provide additional oversight but primarily help a test engineer organize and plan the specifics of each test case before committing to code or formal definition. These types of documents address the planning aspects of a testing project.

Full development of these documents is beyond the scope of this TxDOT research project. In the case of the CCTV test procedures, the researcher capitalized on test procedures already in the public domain, and upfront planning did not appear to be necessary. For the traffic signal controllers, the researcher had a test case specification from a previous project that was appropriate. A previous test design specification was somewhat appropriate, and the researcher believes that an update that references the TxDOT department material specification DMS 11170-TSC, Fully Actuated, Solid-State Traffic Signal Controller Assembly provides useful information (*2*). Appendix F contains a Test Design Specification and a Test Case Specification for addressing all the functions of a traffic signal controller. It uses IEEE Std. 829 – IEEE Standard for Software Test Documentation as a guide on the organization and contents.

A number of state departments of transportation are adopting International Organization for Standardization (ISO) 9000 in order to improve quality (*21*). The following test design specification and test case specification are two types of document examples that would satisfy some of ISO 9000 requirements.

Traffic Signal Controller Test Procedures

Using the information in Appendix F as the basis for development, the researcher provides the details of a limited set of test cases and their test procedures in Appendix G. The following traffic signal controller test cases focus on:

- PRL Information
- Four-Phase Diamond Sequencing
- Four-Phase Diamond Detector Operations
- Global Configuration
- Security

Some additional test cases also look primarily at the functionality. These test cases are based on user needs in that they look at what services a user would expect or perform on a conformant device. These test cases consist of:

- Managing Phase Configuration and Initialization
- Retrieving Phase Status Information
- Retrieving Manufacturer Information
- Setting Up and Retrieving Log Data
- Retrieving System Status Information
- Setting Up and Executing Timebase Events
- Setting Time and Date

Traffic Signal Controller Test Results

The suggestion for reporting traffic signal controller test results is to use the same method as illustrated for CCTV. The PRL from the NTCIP 1202-ASC standard serves as a template and the ASC PRL Information prequalification test script (see Appendix G) outputs the results onto that form (*16*). The completed test results would be similar to what appears in Appendix D. In the case of other test procedures, the NTCIP test procedure itself serves as the template and the related test script transcribes the results onto the template.

The researcher's definition of the traffic signal controller test procedures do not follow the method as prescribed in NTCIP 8007 – Testing and Conformity Assessment Documentation within NTCIP Standards Publications (NTCIP 8007-TEST) (*22*). After reviewing the result reports for the CCTV test procedures, the researcher believes that showing results for every test step does not enhance readability and may lead to overlooking something important. For example, Figure 2 shows "Pass" for every test step. It may be clearer to show only "Pass" for the critical steps as shown in Figure 3. Any test step that does fail would show a "Fail," and the rest of the indications in the test procedure template would be unchanged.

Test Step Number	*Test Procedure*	*Results*
1	SET labelText.<*alarmCLabIndex*> = <*alarmCLabText1*> labelHeight.<*alarmCLabIndex*> = <*alarmCLabHeight1*> labelColor.<*alarmCLabIndex*> = <*alarmCLabColor1*> labelStartRow.<*alarmCLabIndex*> = <*alarmCLabStartRow1*> labelStartColumn.<*alarmCLabIndex*> = <*alarmCLabStartColumn1*>	
2	SET alarmLabelIndex.0 to <*alarmCLabIndex*> 00 00 00 00 00 00	
3	USER VERIFY that no labels are being shown.	Pass
4	SET alarmLatchClear.0 to 0x00	
5	Turn on the alarm and USER VERIFY the label for the alarm is shown.	
6	GET alarmStatus.0 and alarmLatchStatus.0	
7	VERIFY RESPONSE VALUE alarmStatus = 0x80 alarmLatchStatus = 0x80	Pass
8	SET alarmLatchClear.0 to 0x00	Pass

Figure 3. Test Results Indicating Critical Results.

DETECTOR REQUIREMENTS

During the course of developing test procedures for the four-phase detector operations as defined in DMS 11170-TSC, the researcher ran across a situation where the requirements may not be addressing an operational need (*2*). The description of how detectors 1 and 5 are to operate does not address the situation in which there is a call for service on one of the left turns when a controller is resting in the opposing through green. Signage and the physical geometry of an actual intersection may explain it, but when an implementation follows the requirements, a call does not register.

CHAPTER 3:
DEVELOPING ADDITIONAL TEST PROCEDURES

INTRODUCTION

In order to provide TxDOT with a sense of the effort one needs to develop test procedures, the researcher presents Table 5. The table references the standards that would require test procedures, statistical information about the number of object definitions and tables to gauge complexity, information about NTCIP efforts to develop test procedures, and the researcher's estimates to implement test procedures in a suitable TxDOT testing environment. The number of object definitions and tables comes from Version 1 of each standard. Typically, Version 2 of each standard contains more objects.

The estimates come from experience in deriving the test procedures in this report, previous test procedure development in other projects, and conversations with several NTCIP working group chairs, technical editors, and/or consultants. The test procedures under consideration are not exhaustive. For example, virtually any object can trigger the logging of an event. Rather than test all possible triggers, the procedures would look at typical objects of each data type to gauge overall correctness of the function.

In general, read-only objects represent either preset values or status information with status information correlating to the overall complexity. Preset values typically define the number of instances associated with entries in a table. A test procedure to check presets is trivial. Status information, on the other hand is relatively complex to test. This testing usually involves the setup of conditions to invoke individual states of a status object. The number of status objects is approximately equal to the number of read-only object definitions minus the number of tables.

Table 5. NTCIP Standard Statistics and Test Procedure Efforts.

Standard	Object Definitions			Tables	NTCIP Test Procedures	Effort for Test Procedures
	Read-Only	Read-Write	Total			
1103-TMP	77	35	112	7		4-6 months
1201-GLO	46	27	73	8	Planned	2-3 months
1202-ASC	85	98	183	21		8-10 months
1203-DMS	79	53	132	7		2-4 months
1204-ESS	83	7	90	4	Planned	2 months
1205-CCTV	35	54	89	2		2.5 months
1206-DCM	151	79	230	30		8-10 months
1207-RMC	59	143	202	19		5-7 months
1208-SW	22	37	59	8		3 months
1209-TSS	27	18	45	8		4-6 months
1210-FMS	77	104	181	23		7-9 months
1211-SCP	43	5	48	4	Planned	2 months
1213-ELMS	26	63	89	9		3 months

ITS FIELD DEVICE ESTIMATES

The following sections provide a description of the individual estimates. The estimates do not include any effort to locate and acquire an implementation in order to validate the test procedures. One should also understand that additional hardware is often necessary to check the functionality of an implementation. For example, a data collection and monitoring device would need various types of sensors to produce valid or runtime-status information values. In the case of signal control and prioritization, a system consists of a request generator, request server, and traffic signal controller. All three components would need to be on hand to fully test a system.

NTCIP 1103-TMP

The NTCIP 1103 Transportation Management Protocols (NTCIP 1103-TMP) standard contains statistical objects related to STMP and SFMP, STMP configuration objects, the event and report objects, and the community name objects (*23*). The effort to define test procedures for the STMP statistical objects is moderate because it requires the creation of error conditions. Although some related procedures exist, SFMP procedures will require design time. Complicating the development of SFMP procedures is the fact that testing tools do not support the protocol, and implementations are not apparent. Therefore, SFMP test procedures are not part of the estimate. The estimate for the level of effort for STMP configuration objects is relatively low because there has been preliminary development. This estimation assumes that the procedures will address the encodings for only a sample of the various object types. A more robust procedure that looks at several dozen possible typical messages would take considerably longer. The effort for event and report objects test procedures is also relatively low because there has been preliminary development. The trap management object that will appear in NTCIP 1103-TMP Version 2 would double the effort and push the estimate out to the 6-month mark (*23*).

NTCIP 1201-GLO

The NTCIP 1201 – Global Object Definitions (NTCIP 1201-GLO) standard contains objects that relate to general functions within ITS field devices (*7*). It covers non-device specific functions like time and date, auxiliary input and output, and general scheduling information. The effort for global objects procedures is relatively low because there has been some preliminary development. There is a strong likelihood that an NTCIP working group will add these to the standard itself. The researcher estimates the effort for developing NTCIP 1201-GLO test procedures at 2 to 3 months.

NTCIP 1202-ASC

The effort to develop test procedures for NTCIP 1202 – Object Definitions for Actuated Signal Controller Units (NTCIP 1202-ASC) ranges from 8 to 10 months (*16*). There are private industry efforts to develop NTCIP 1202-ASC test procedures. However, there is some reluctance to place these procedures in the public domain. A proposal to add them to the NTCIP standard was not approved. Assuming these private industry procedures are not available, then the effort will be somewhat lengthy. This estimate assumes a limited number of test cases wherein the number of combinations and permutations is minimal. The estimate includes 1 month to address three-phase diamond and dual four-phase operations. With these assumptions, the estimate for NTCIP 1202-ASC test procedures is between 8 and 10 months.

NTCIP 1203-DMS

The estimate for NTCIP 1203-DMS test procedures ranges from 2 to 4 months. The unknown variable is support for Version 2 *(5)*. Test procedures for Version 1 are already in the public domain. Assuming these test procedures are acceptable and they simply need updating and reformatting, then the effort is about 2 months. From a conversation with the editor of the NTCIP-1203 standard, Version 2 test procedures may be currently under development as part of a Virginia DOT project *(5)*. If these are to serve as the basis for TxDOT procedures, the effort estimate is 4 months.

NTCIP 1204-ESS

For NTCIP 1204 – Environmental Sensor Station Interface Standard (NTCIP 1204-ESS), the estimate is 2 months *(24)*. In a conversation with the NTCIP program manager on August 10, 2006, the NTCIP 1204-ESS working group has a notice-to-proceed with adding test procedures to the standard *(24)*. Assuming these are acceptable to TxDOT, then the effort will be translating them into appropriate scripts. The researcher's estimate for NTCIP 1204-ESS test procedures is 2 months.

NTCIP 1205-CCTV

This research report includes a set of test procedures for NTCIP 1205 – Object Definitions for Closed Circuit Television (CCTV) Camera Control. Omitting the effort to locate and acquire NTCIP compliant hardware to validate the procedures, the effort to reformat the Enterprise test procedures into the NTCIP 8007 – Testing and Conformity Assessment Documentation within NTCIP Standards Publications (NTCIP 8007-TEST) format and convert them to SimpleTester™ [21] for NTCIP test scripts was approximately 2.5 months *(22)*.

NTCIP 1206-DCM

NTCIP DCM working group chair's estimate for test procedures related to NTCIP 1206 – Object Definitions for Data Collection and Monitoring (DCM) Devices (NTCIP 1206-DCM) is between 5 and 7 months *(25)*. Given the number of status information objects, however, this appears to be conservative. The researcher believes that describing and defining the test

---

[21] SimpleTester™ is a trademark of SimpleSoft, Incorporated, Mountain View, California.

conditions will take considerable effort. Even without a working knowledge of a DCM device, the number of status information objects is almost double that of other devices with a high level of complexity. For this reason, the researcher's estimate is 3 months longer than the chair's estimate and is put at 8 to 10 months.

NTCIP 1207-RMC

The estimate for NTCIP 1207 – Object Definitions for Ramp Meter Control (RMC) Units (NTCIP 1207-RMC) comes from the NTCIP RMC working group chair *(26)*. When one compares the number of status information objects to other standards, this estimate is somewhat high. However, in the case of RMC devices, the researcher defers to the judgment of the working group chair whose estimate is 5 to 7 months.

NTCIP 1208-SW

Given the least number of status information objects of any standard, the estimate for NTCIP 1208 – Object Definitions for Closed Circuit Television (CCTV) Switching (NTCIP 1208-SW) is put at 3 months *(27)*.

NTCIP 1209-TSS

The NTCIP 1209 – Data Element Definitions for Transportation Sensor Systems (NTCIP 1209-TSS) estimate is between 4 and 6 months *(28)*. The number of objects in Table 5 comes from Version 1 of NTCIP 1209-TSS. This number of objects is at the lower end of the scale when considering other NTCIP standards. However, the working group chair cautions that Version 2 is going to include additional object support for machine vision and vehicle classification. Since these two areas are of interest to TxDOT, the upper end of the 4 to 6 month estimate is for support of Version 2.

NTCIP 1210-FMS

The basis for the estimate of NTCIP 1210 – Field Management Stations - Part 1: Object Definitions for Signal System Masters (NTCIP 1210-FMS) test procedures is the current working group draft *(29)*. Even though the final object count will likely change before Version 1 has full approval, the researcher and the FMS working group consultant are in general agreement on the amount of effort to develop a set of test procedures. The estimate for FMS test procedures is 7 to 9 months.

NTCIP 1211-SCP

Even though the NTCIP 1211 – Object Definitions for Signal Control and Prioritization (NTCIP 1211-SCP) standard deals with the potential for a system consisting of multiple physical entities, the effort to implement a set of procedures for TxDOT should be relatively small (*30*). This is because an NTCIP project plan to add them to the standard has approval. If these NTCIP test procedures meet with TxDOT's approval then the effort will be translating them into appropriate scripts. This estimate is 2 months.

NTCIP 1213-ELMS

The estimate for NTCIP 1213 – Objects Definitions for Electrical and Lighting Management Systems (NTCIP 1213-ELMS) is 3 months (*31*). This estimate comes from the number of object definitions in the standard and a general discussion with the working group chair. Given the small number of status information objects, defining the procedures should be relatively easy. As stated in the introduction, however, the bigger issue will be to identify and acquire field devices to verify the procedures.

CHAPTER 4:
APPLYING TESTING PROCEDURES

TESTING PROCESSES

TxDOT documents and specifications describe two testing processes. Tex-1170-T – Sampling and Environmental Testing of Traffic Signal Controller Assemblies: Traffic Signal Controllers and Conflict Monitors (Tex-1170-T) defines an internal TxDOT process that applies to traffic signal controllers and cabinets *(32)*. The second process comes from Special Specification 6504 – Testing, Training, Documentation and Warranty (SS 6504) *(33)*. This specification describes a testing process that is the responsibility of a contractor to perform. The following headings describe the two processes and apply different testing techniques to each of them. A draft information report, NTCIP 9012 – Testing and Conformity Assessment User Guide for NTCIP Field Devices and Center-to-Field Communications (NTCIP 9012-TG) provides general background and guidance to an agency defining a testing process/program for testing devices that incorporate NTCIP Standards *(34)*.

INTERNAL TxDOT TESTING PROCESS

Figure 4 illustrates the process in Tex-1170-T. For a bidder to provide equipment under a contract, the equipment must usually be on TxDOT's Qualified Products List. To qualify for the list, equipment must pass QPL testing. Assuming that it is on the list, a bidder provides equipment that meets any contract-specific specifications or provisions to the district office. The district sends a sample of the equipment to the Traffic Operations Division for environmental and, possibly, QPL testing. This round of QPL testing only takes place if the sample is not already on the QPL list. At the same time, the district checks and configures the equipment for its specific installation requirements. Once the equipment passes these checks, it undergoes a field check to see if it operates properly in the system.

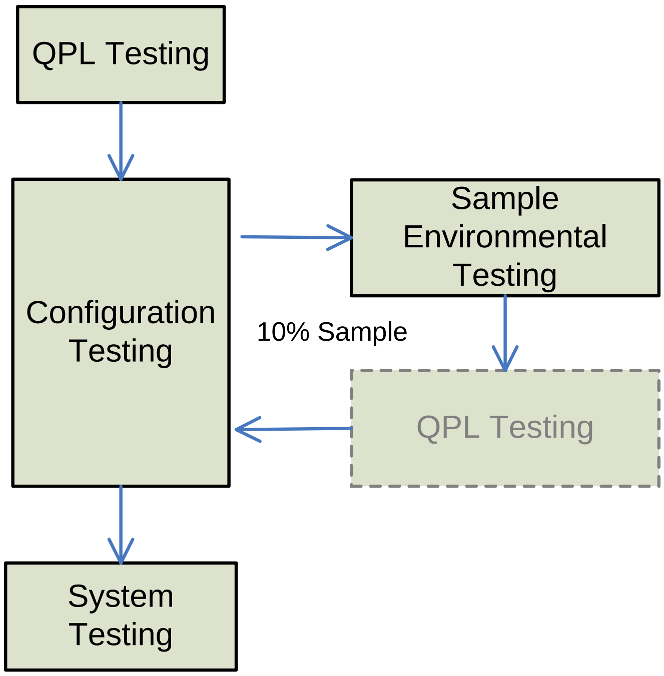


Figure 4. TxDOT Testing Activities.

Prequalification Testing

Prior to performing QPL testing, a tester should review a manufacturer's completed Protocol/Profile Implementation Conformance Specification (PICS) or perform the Protocol/Profile Requirements List Information Test Procedure in Appendix C to produce a PICS. NTCIP standards that deal with object definitions for a device usually provide a PRL that has one or more tables summarizing object definitions in the Management Information Base. After filling out a PRL with the information about support for objects or features, indicating the range of supported values, and showing values for indexed items, it becomes a PICS. Using the PRL/PICS as the basis for a report produces a uniform and consistent manner in which to compare similar devices.

The script of the PRL test procedure in Appendix C creates a PICS for a CCTV control unit. The test procedure creates the PICS by:

- retrieving minimum project requirements values and indexing parameters,
- checking that all objects in a MIB can be read,
- performing a set operation on writeable objects with a sampling of values to check an object's range, and
- recording the information on a PRL that appears in the NTCIP standards.

A SimpleTester™ for NTCIP script automates the process. The script for CCTV takes only several minutes to run. Appendix D shows a sample completed PICS.

A review of a PICS serves the purpose of prequalification testing or one of the first steps in QPL testing. One can use the PICS to determine if it is worth spending the time to test an implementation extensively. If the PICS indicates that an implementation does not support a required function or the required number of instances of an object, testing the functions and objects that are implemented would not serve any purpose.

QPL Testing

The focus of QPL testing is compliance to TxDOT specifications for a specific device. Compliance to the specification entails a 100 percent check of all requirements. It is exhaustive and covers:

- hardware design,
- conformance to external standards,
- functionality, and
- documentation.

QPL testing uses a device's specification as a guide to ensure that equipment meets all of the requirements. While testing for compliance to NTCIP does not address hardware design and documentation, it can address conformance to some external standards and functionality. Compliance to TxDOT specifications does not necessarily have to be a separate process. Consider the Absolute Pan Motion test procedure in Appendix C that tests one of the panning motion functions defined in NTCIP 1205-CCTV. The procedure consists of:

1. Move the camera to a predefined position.
2. Ask the user to verify that the camera moved to that position.
3. Internally verify that the camera moved to that position.
4. Move the camera to another predefined position.
5. Ask the user to verify that the camera moved to that position.
6. Internally verify that the camera moved to that position.

The intent of the procedure is to check the conformance of three NTCIP objects. As it is stated, the procedure could be suitable for all the phases of the TxDOT testing process. By referencing externally defined positions, the procedure is suitable for statewide, district, and one-time use because the positions values are customizable to the individual requirements. The means by which the user verifies the new positions could also customize it to TxDOT QPL or to configuration and system testing. In the case of QPL, one could quantitatively measure the angle. For example, during QPL, one could use a protractor to define reference points to measure angular positions. For configuration and system testing, simply estimating the angle may be sufficient.

Configuration Testing

The focus of configuration testing is on meeting project-specific requirements. Using an approach where variables come from an external source allows a tester to customize the test procedure to his or her needs. Using the CCTV Absolute Pan Motion test procedure, the test could use pan position values that come from the project specification or the geometry of the intended location.

During configuration testing, one common practice is to run a test repetitively to simulate actual field operations and stress the unit. In this case, a simple modification to the procedure might be to insert additional looping instructions. Using the CCTV Absolute Pan Motion test procedure as an example, the procedure would look something like the following:

1. For N = 1 to predefined reiterations
    1. Move the camera to a predefined position
    2. If predefined reiterations = 1 then
        i. Ask the user to verify that the camera moved to that position
    3. Internally verify that the camera moved to that position
    4. Move the camera to another predefined position
    5. If predefined reiterations = 1 then
        i. Ask the user to verify that the camera moved to that position
    6. Internally verify that the camera moved to that position
2. Next reiteration

Defining the number of predefined reiterations as 1 makes it suitable for QPL testing where a tester physically verifies the positions. Defining the number of predefined reiterations as 100 could make it more appropriate for configuration testing where the emphasis is on stressing the equipment.

Sample Environmental System Testing

During environmental testing, a test chamber subjects a device to temperature and line-voltage extremes that may occur in actual field operation. The device should continue to run and time intervals correctly under combinations of high and low environmental conditions. Although NTCIP test procedures are not directly applicable to environmental tests, the browser functions of one of the active testing tools could find usage in changing the configuration of a device without having to enter the test chamber. A test procedure that checks a traffic signal controller's NTCIP 1202-ASC download features could configure a controller with a standard set of configuration controls and interval times *(16)*. There is nothing to prevent any NTCIP functional test procedure from also being run while a device is in an environmental chamber.

System Testing

During system testing, a management application connects to a device either while the device is still at a facility or after installation in the field location. NTCIP test procedures have two uses during system testing. They can serve as a test of the management application to show that it supports the functionality expressed by the NTCIP objects. An instrumentation testing tool can also validate that a device responds correctly even though communications take place over an agency's communications infrastructure.

CONTRACTOR TESTING PROCESS

Figure 5 illustrates the process in SS 6504. During design approval testing, the contractor either runs environmental tests directly or has an independent testing laboratory conduct them. During demonstration testing (conducted prior to installation), the contractor performs a physical inspection of the equipment and performs operational tests to ensure compliance to the specifications *(33)*. After the contractor installs the equipment but before connection to any other components of the system, stand-alone testing verifies functional operations. After connection, system integration testing demonstrates that all control and monitor functions are operating properly. TxDOT personnel do not perform the tests, but a reserve clause in the specification allows someone from TxDOT to observe the tests. The TxDOT engineer is responsible for overall approval and final acceptance.

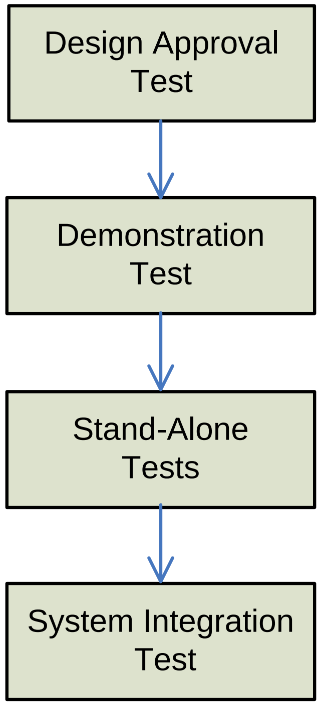


Figure 5. Contractor Testing Activities.

NTCIP test procedures find application in contractor testing in much the same way as they do in the internal TxDOT testing process. NTCIP test procedures can add standardized functional test procedures during the design approval testing. During the operation-testing portion of the demonstration test, the NTCIP test procedures cover the functionality expressed by the NTCIP object definitions. The NTCIP object definitions, in turn, cover most, if not all, the functionality of a device. During the stand-alone test, the NTCIP test procedures provide not only standardized tests but also a means to conduct the testing. Although NTCIP standards do not formalize a set of test procedures as yet, the de facto procedures are undergoing peer review. With testing tool support coming from many sources, contractors do not have to develop their own software to conduct the test. There is also the benefit that by prescribing the use of NTCIP procedures, the engineer gains a greater understanding of what is taking place. During system integration testing, most of the communications driving the functionality of a device come from a management application. The NTCIP test procedures would not apply in this case. However, the instrumentation testing tools that support NTCIP could provide an independent means to verify that a management application is issuing the proper commands and that the device is responding correctly. SS 6504 also requires that contractors submit test procedures and data forms to the TxDOT engineer prior to any actual testing for his or her approval. NTCIP test procedures provide more consistency in the documentation and are easier to understand (*33*). Adopting the reporting format as suggested in this research should make it easier for engineers to interpret the report and make the reports consistent across all contractors.

A TxDOT engineer or other TxDOT personnel could verify contractor testing by the use of an instrumentation-testing tool. This type of testing tool can provide independent verification that a contractor is performing the test procedures as described and that the results are as they should be. The same information would also be available when used during system integration or actual system operation. An instrument-testing tool provides a message view of exchanges between two parties. The tool can show what a management or test application sends. It will also show the responses from the device under test. Rather than showing a string of bytes that appear on the wire, a tool can decode the bytes into human readable parameters that indicate the type of command sent, the names of NTCIP objects involved, and the values of the objects. Figure 6 is an example of how one instrument-testing tool presents information to a user.

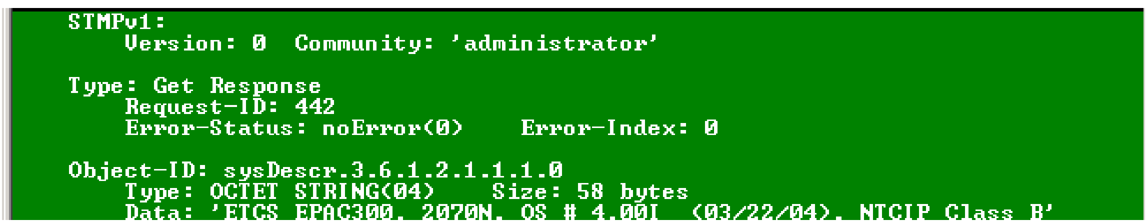


Figure 6. Instrumentation Testing Tool Information Example.

The display shows the response of a traffic signal controller to a get request asking for a description of the device. The response shows that the STMPv1 protocol is used, the community name or access code was "administrator," the packet type was a "get response," and there were no errors. The name of the NTCIP object that contains the description is "sysDescr" and that the value is "ETCS EPAC300, 2070N, OS …" Those familiar with traffic signal controllers will recognize this as an Eagle Traffic Control System – EPAC 300 series signal controller running on ATC 2070N hardware.

CONFIGURATION MANAGEMENT AND VERSION CONTROL

As manufacturers implement the NTCIP standards and users deploy equipment in the field, the standards development groups receive feedback about problems and requests for enhancements. This invariably leads to a revision of the standards. Unless one takes specific steps to avoid the situation, it is possible to have systems consisting of equipment conformant to different versions of a standard. For example, the initial attempts to define object definitions related to time-of-day resulted in several alternate but valid ways of how to treat the objects. The dates for implementing daylight saving time are now different. Revisions to the standard reflect corrections and the new rules. Some manufacturers are implementing the newest versions of the standard. Although NTCIP strives to make revisions backwards compatible, it is not always possible. Unless a management application is up-to-date, it will not be able to deal with any changes.

Any specification that references an NTCIP standard should specify the specific version and revision date. Since some NTCIP standards deal with functionality applicable to all field equipment (NTCIP 1201-GLO, for example), specifications of various equipment may need to be consistent in this regard as well (*7*). If one specification calls for a specific version number, interoperability problems may crop up if other specifications reference another version. An agency should also maintain records containing location and version information. These two steps will minimize incompatibility problems and help in understanding the impacts of any future upgrade.

CHAPTER 5:
TRAINING

INTRODUCTION

This chapter presents two training course outlines. The first looks at testing from an NTCIP perspective and the second outline looks at testing from a TxDOT perspective. The first outline begins with an explanation of the difference between conformance testing in relation to the NTCIP standards and compliance testing in relation to TxDOT specifications. The outline then addresses background information on the NTCIP standards in order to put testing into perspective. There is a discussion on the two types of standards: data dictionaries and protocols/profiles. The background also covers two NTCIP standards related to testing, and there is also a reference to the NTCIP framework on how to combine standards to build an implementation. The next part of the first outline deals with terminology and techniques. The last part covers how to interpret results. Different groups and organizations have different methods for reporting test results. Testing tools also incorporate some type of report. If a large agency, such as TxDOT, needs to examine these reports, it takes a bit of understanding to draw meaningful conclusions.

The second outline looks at testing from the TxDOT perspective. Given enough time and resources, one could fully test everything covered by NTCIP and the functionality associated with the object definitions. However, there is usually a lack of resources, and time is always at a premium. These limitations introduce the subject of risk management. Risk management looks at techniques to minimize the amount of testing and still maintain a high level of confidence that an implementation is correct. The outline then looks at what to test and the general techniques to use. The next part examines various tools that are available. It explains the three types of tools: active, emulator, and instrumentation and covers some of their characteristics. The last part of the outline discusses configuration management. Without specifying specific standards versions, one can expect interoperability issues. The chapter ends with a suggested evaluation form.

AUDIENCE

The level of detail in this training course is meant for someone responsible for planning or carrying out testing activities. Most of the material is non-technical in nature. However, a person may need to ensure that messages use the proper formatting, encoding, and NTCIP protocols. This level of understanding may be especially important for someone monitoring contractor testing or trying to isolate faults during system testing, as well as for someone evaluating a testing tool. To provide a sufficient level of understanding, there is a technical discussion about the fields of an encoded message.

TRAINING CLASS OUTLINES

1. Testing from the NTCIP Perspective

1.1. Introduction

1.1.1 Conformance testing versus compliance testing.

1.1.2 Conformance to NTCIP standards.

1.1.3 Compliance to TxDOT specifications.

1.2. Background Information

1.2.1 NTCIP standards promote interoperability, interchangeability, and compatibility. Interoperable is desirable so that system components from different vendors can work together. Interchangeable is desirable so that there is no loss in functionality when replacing system components with similar components from different vendors. Compatible is important so that system components can share a common communications infrastructure.

1.2.1.1 Data Dictionaries define the words, and there is one dictionary for each field device.

1.2.1.1.1 Objects / Data elements define parameters, controls, and status.

1.2.1.1.2 Conformance Groups are collections of objects that together perform some specific function or task.

1.2.1.1.3 Management Information Base (MIB) is a collection of objects related to field device. A MIB can come from multiple dictionaries.

1.2.1.1.4 Protocol/Profile Requirements List (PRL) is a checklist of object support.

1.2.1.1.4.1 PRL Object Types indicate what type of tests may apply to the objects.

1.2.1.1.5 Requirements Traceability Matrices provide a mapping between user needs/requirements and objects that address them.

1.2.1.2 Communications Protocols define the rules for combining the words and transmitting them over the media.

1.2.1.3 Application-level communications protocols handle interface between end-application (e.g., Signal Controller) and transport-level protocols.

1.2.1.3.1 Simple Network Management Protocol (SNMP) commands are Get, Set, GetNext, and Trap and it uses Basic Encoding Rules (BER) encoding rules. The NTCIP Guide has several examples of SNMP encoded messages.

1.2.1.3.2 Simple Transportation Message Protocol (STMP) commands are Get, Set, and GetNext but it uses Octet Encoding Rules (OER) encoding rules and is limited to 13 predefined messages. The NTCIP Guide has an example of STMP encoded messages.

1.2.1.3.3 Simple Fixed Management Protocol (SFMP) is the same as SNMP but Object Identifiers (OIDs) use a different node as a reference.

1.2.1.3.4 File Transfer Protocol (FTP) is the same as that used on the Internet.

1.2.1.3.5 Trivial File Transfer Protocol (TFTP) is the same as FTP but without all the directory commands and other features.

1.2.1.4 Transport-level communications protocols handle end-end connections and transfers as well as routing through networks.

1.2.1.4.1 Transportation Transport (T2) Profile is for non-networked environments where throughput is a concern.

1.2.1.4.2 User Datagram Protocol (UDP) is for a networked environment with best effort delivery.

1.2.1.4.3 Transport Control Protocol (TCP) is for a networked environment with guaranteed delivery.

1.2.1.4.4 Internet Protocol (IP) is for a networked environment and handles routing.

1.2.1.5 Subnetwork level protocols handle point-to-point connections and deal with issues related to putting information on media and errors.

1.2.1.5.1 Point to Multi-Point Protocol (PMPP) uses rules that are similar to the one that people use to dial into an internet service provider like AOL but supports party lining. There are separate standards for RS232 and FSK modems.

1.2.1.5.2 Point-to-Point Protocol (PPP) is the one that people use to dial into an internet service provider like AOL. PPP specifies the Challenge Handshake Authentication Protocol (CHAP) for authentication.

1.2.1.5.3 Ethernet Protocol is the same one used on office computers. It provides high throughput and robustness.

1.2.1.5.4 The NTCIP Guide has several examples of message encoding over various protocols.

1.2.1.6 NTCIP standards do not define a standard method of encrypting information. User access “passwords” are visible on the wire.

1.2.2 Other Standards

1.2.2.1 NTCIP 8007 – Testing and Conformity Assessment Documentation within NTCIP Standards Publications (NTCIP 8007-TEST) prescribes format of test procedures using natural language to describe test steps. The language includes basic keywords.

1.2.2.2 NTCIP 9012 – Testing and Conformity Assessment User Guide for NTCIP Field Devices and Center-to-Field Communications (NTCIP 9012-TG) provides general testing information in the context of ITS.

1.2.3 Implementations

1.2.3.1 An implementation consists of a combination of four levels of standards: information, application, transport, and subnetwork.

1.2.3.2 The MIB that defines an implementation comes primarily from the information-level data dictionary standards but can include objects related to communications level – application, transport, and subnetwork standards.

1.3. Testing Terminology and Techniques

1.3.1 Device Under Test (DUT) is the implementation that is undergoing the test.

1.3.2 Positive Testing exercises a DUT in a manner that is consistent with its normal operating conditions.

1.3.3 Negative Testing (a.k.a. Error Seeding) exercises a DUT in abnormal operating conditions such as out-of-range variables, input errors, and fault conditions.

1.3.4 Sampling is a technique that tests only a portion of the units with the assumption that the others will perform in a similar manner.

1.3.5 Regression is the retesting of a previously tested program following modification to ensure that faults have not been introduced.

1.3.6 Black-box Testing is based on an analysis of the specification of the component without reference to its internal workings as in White-box Testing.

1.3.7 Boundary Value Analysis is a selection of test values that surround the “boundaries” of a parameter’s input range. Choices often include maximum, minimum, and trivial values.

1.3.8 Stress Testing subjects a system to incorrect, abnormal, or unrealistic inputs or conditions with the intention of producing a failure. It looks at testing at or beyond the limits of its specified requirements.

1.3.9 Validation and Verification (V&V) - Verification is testing that determines whether an implementation is built correctly whereas validation testing checks whether the correct implementation was built.

1.4. Interpreting Results

1.4.1 NTCIP PRLs standards provide Yes / No for each object or function and do not provide any additional guidance.

1.4.2 NTCIP test procedures use a Pass / Fail method and do not provide any additional guidance other than a test case basis.

1.4.3 Battelle’s suggestion was to evaluate “critical” functions.

1.4.4 Non-TxDOT Test Procedures

1.4.4.1 Some Enterprise test procedures use a point scale with a minimum passing value.

1.4.4.2 SimpleTester™ uses a scale of 1 to 4 and scores on a test case basis.

1.4.5 Proposed TxDOT Test Procedures use NTCIP PRLs and test procedure forms.

1.4.5.1 User determines what constitutes whether DUT is suitable.

1.4.6 Contractor Testing is unknown but likely varies by contractor.

2. Testing from the TxDOT Perspective

2.1. Risk Management

2.1.1 Do you have to try all possible values in an object acceptable range of values?

2.1.2 Do you have to check all instances of an object if there are 16 duplicates of the same thing?

2.1.3 If a DUT supports all the objects in the specifications but does not support some object required by NTCIP, is it still acceptable to use?

2.1.4 If a DUT supports all the values of an object required for a project but fails on some values that are called for in the specifications, is it still acceptable to use?

2.1.5 Do all possible permutations and combinations need to be tested? For example, in a traffic signal controller, is a detector call for service on another phase entered when the current phase is green, yellow, and red? Does the call register when a traffic signal controller is in red rest?

2.2. What to Test

2.2.1 Ensure that all required objects are readable.

2.2.2 Check whether value of objects that represent some limit or number of instances meets or exceeds a required value.

2.2.3 Ensure that parameter objects accept the required range of values.

2.2.3.1 Perform boundary analysis with positive and negative range.

2.2.3.2 Perform sampling of mid-range values.

2.2.3.3 Perform sampling of multiple instance parameters.

2.2.4 Ensure that control objects can be set to the required range.

2.2.4.1 Goal is to test 100% of all values but tempered with realistic permutations and combinations.

2.2.5 Verify that all status objects return the appropriate value when parameter and control objects are set accordingly.

2.2.5.1 Goal is to check 100% of all values by creating scenarios that produce status values.

2.2.6 Perform regression testing when software changes.

2.3. Testing Tools

2.3.1 Active tools simulate the operation of management application.

2.3.1.1 Tools range from device-specific, canned testers to fully customizable, general-purpose tools.

2.3.1.2 Cost runs from free to mid four figures.

2.3.1.3 One should use caution in that it is best to verify that a tool does what it says it does.

2.3.2 Emulator tools act like field devices.

2.3.2.1 Tools range from device-specific, canned emulators to general-purpose that can be customized to simulate the entire system consisting of many different types of field devices.

2.3.3 Instrumentation can monitor and analyze information exchanges.

2.3.3.1 Instrumentation can act as arbitrator when using other tools and during systems testing. Project engineer can use it to verify testing by contractors.

2.3.4 High-ended active and emulator tools have the ability to run user-defined scripts (programs) that define tool operation.

2.3.4.1 Scripts are generally proprietary but one set of tools uses the Tool Command Language (TCL), a common language in many testing and simulation applications.

2.3.4.2 Some tools allow interface to other programs (e.g., hardware-in-the-loop).

2.3.4.3 Once procedures are written and verified, the process of testing is greatly simplified.

2.4. Configuration Management

2.4.1 Specify NTCIP standard version numbers.

2.4.1.1 Mentioning “or latest version” can lead to interoperability issues.

2.4.1.2 Management application and field devices should support same version.

2.4.2 Configuration management consists of identification, change control, configuration auditing, and reporting.

2.4.2.1 Identification involves enumerating what is currently being used to establish baseline and describing new versions as to applicability and what is changed.

2.4.2.2 Change control involves understanding the impact on performance reliability and compatibility.

2.4.2.3 Configuration auditing involves identifying what components will need updating.

2.4.2.4 Reporting involves making sure that everyone involved is kept informed.

TRAINING CLASS EVALUATION FORM

NTCIP Testing Training Course
Evaluation Form

Location: ____

Date: ____

Your Agency: ____

Your Position: ____

Course Content

		Yes	Somewhat	No
1.	Did the course meet your expectations? Comments:	1	2	3
2.	Was the material presented at the correct level of difficulty? Comments:	1	2	3
3.	Were the presentation and guidebook appropriately geared to providing you the information you needed? Comments:	1	2	3
4.	Do you feel the time spent on this course was beneficial? Comments:	1	2	3

General Observations

5. What did you like most about the course?

6. What did you like least about the course?

7. What can we do to improve this course in the future?

8. Do you have any other suggestions or comments?

Thank you for taking the time to complete this course evaluation form. Please make sure the course instructor receives it before you leave.

CHAPTER 6: RECOMMENDATIONS

TESTING FRAMEWORK

The following are additional researcher's recommendations on defining a framework for the testing of conformance to NTCIP and integrating it into the current TxDOT testing program.

1. Review the detector operation requirements for detectors 1 and 5 with respect to the four-phase diamond operation that appears in DMS 11170-TSC. When the opposing through movement is green, a call for service on these detectors does not register.
2. Reorganize special specifications titled National Transportation Communications for ITS Protocol for Field Equipment so that they distinctly identify information, application, transport, and subnetwork-level profiles and protocols requirements. Along with this, consider whether the application, transport, and subnetwork-level profiles and protocols requirements can be stand-alone documents.
3. Add wording in any specification that relates to NTCIP to address application, transport, and subnetwork-level object conformance groups. Support for the object definitions could help when experiencing communications problems.
4. Consider whether any planning for test documentation should follow the IEEE Standard for Software Test Documentation. This would help in any transition to ISO certification.

FUTURE DEVELOPMENT

ELMS Test Procedures

The NTCIP 1213-ELMS standard has progressed to the stage where it includes a set of dialogs that illustrate the exchange of information between a management application and a field device *(31)*. Text also describes how a management application would carry out the exchange of information. What it lacks, however, is a standard set of procedures to verify a correct implementation. There are no plans within the NTCIP development process to add test procedures to the standard. Even though TxDOT representatives made significant contributions to the development of the NTCIP ELMS standard, the TxDOT specifications for ELMS do not reference any NTCIP requirements.

From the findings of this research project, the process and steps to follow for the development of ELMS test procedures are in place. A research or implementation project that focuses on ELMS would provide templates for TxDOT documentation that specifies NTCIP requirements. A set of test procedures would create a ‘conformant management station’ as defined in the NTCIP-1213 ELMS. A conformant management station would provide TxDOT personnel with the means to test for conformance to the NTCIP standards and test for compliance to the TxDOT specifications.

Generic Database

In the course of developing the four-phase diamond detector operations test procedures for traffic signal controllers, it was necessary to load numerous timing parameters and set various controls in order for the controller to perform as expected. In some cases, the loading and setting involved four parameters for each of 16 phases. When considering other test procedures for traffic signal controllers, there may be hundreds of parameters to be set in order to create suitable test conditions. Other devices have similar complexity.

A test procedure can specify steps to retrieve and save the current parameter values, download appropriate values for the test, and then restore the original values when the test is complete. An alternate approach would be to develop a generic database for the purposes of testing and configuring ITS field devices. The design of the database could use the self-describing information in a Management Information Base that is part of every device standard. A MIB names the parameters, controls, and status information of a device, defines data types and constraints for data entries, provides a description of the data, and describes the structure or organization of data. Organizing this information as a spreadsheet would provide an easy method of entering test values and a convenient way of changing the parameters of a device in a test procedure. This database would create a reusable utility and eliminate the need to hard code test values in a test script.

Personnel at the Florida Traffic Engineering Research Lab (TERL) and at the Idaho National Institute for Advanced Transportation Technology (NIATT) have expressed a need for database utility in their testing activities. One of the AASHTO manufacturer representatives on the NTCIP oversight committee believes that a generic database would also help companies that do have device specific database support in their management application software. Any MIB-based database would never have the ease of use, refined user interface, or understanding of the relationships in the data that manufacturers build into their management application software. Research into the definition of a generic MIB-based database has the potential to simplify the task of defining data sets and providing access to device information when specific management application software is not available.

REFERENCES

1. Special Specification 6025 – CCTV Field Equipment. Published by TxDOT. ftp://ftp.dot.state.tx.us/pub/txdot-info/cmd/cserve/specs/2004/spec/ss6025.pdf. Accessed June 21, 2006.

2. DMS-11170, Fully Actuated, Solid-State Traffic Signal Controller Assembly, Departmental Material Specifications 7-115, Section 19. Published by TxDOT. http://manuals.dot.state.tx.us/dynaweb/colmates/dms/@ebt-link;?target=idmatch(s070019). Accessed July 29, 2005.

3. IEEE Std 829-1998 – IEEE Standard for Software Test Documentation, Institute of Electrical and Electronics Engineers, New York, New York, 1998.

4. TxDOT Specifications, http://www.dot.state.tx.us/business/specifications.htm. Accessed August 21, 2006.

5. NTCIP 1203 – Object Definitions for Dynamic Message Signs (DMS), A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=1203. Accessed August 17, 2006.

6. NTCIP 1205 – Object Definitions for Closed Circuit Television (CCTV) Camera Control, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=1205. Accessed August 17, 2006.

7. NTCIP 1201 – Global Object Definitions, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=1201. Accessed August 17, 2006.

8. NTCIP 2301 – Simple Transportation Management Framework Application Profile, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=2301. Accessed July 25, 2005.

9. NTCIP 2201 – Transportation Transport Profile, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=2101. Accessed August 17, 2006.

10. NTCIP 2101 – Point to Multi-Point Protocol Using RS-232 Subnetwork Profile, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=2101. Accessed August 17, 2006.

11. Special Specification 6026 – National Transportation Communications for ITS Protocol for Dynamic Message Signs. Published by TxDOT. ftp://ftp.dot.state.tx.us/pub/txdot-info/cmd/cserve/specs/2004/spec/ss6026.pdf. Accessed August 17, 2006.

12. NTCIP 9001 – The NTCIP Guide, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=9001. Accessed September 12, 2006

13. NTCIP 2103 – Point-to-Point Protocol over RS-232 Subnetwork Profile, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=2103. Accessed August 17, 2006.

14. Rose, M., and K. McCloghrie, Management Information Base for Network Management of TCP/IP-based internets: MIB-II, RFC 1213, Performance Systems International, Hughes LAN Systems, March 1991.

15. Test Procedures for NTCIP-conformant Closed Circuit Television (CCTV) Camera Controllers, Enterprise Consortium. http://enterprise.prog.org/ (document no longer available). Accessed December 2002.

16. NTCIP 1202 – Object Definitions for Actuated Traffic Signal Controller Units, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=1202. Accessed August 17, 2006.

17. TS 2-2003 – Traffic Controller Assemblies with NTCIP Requirements, National Electrical Manufacturers Association, Rosslyn, Virginia.

18. NTCIP 2104 – Ethernet Subnetwork Profile, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=2104. Accessed June 27, 2006.

19. Travel Time Messaging on Dynamic Message Signs - Portland, OR. Published by the Federal Highway Administration. http://ops.fhwa.dot.gov/publications/travel_time_study/portland/portland_ttm.htm. Accessed July 17, 2006.

20. The City of Palo Alto Uses GarrettCom Ethernet Products in Its Advanced Transportation Management System (ATMS), A GarrettCom Application Note. Published by GarrettCom, Inc. http://www.garrettcom.com/techsupport/appnotes/paloalto_appnote.pdf. Accessed July 17, 2006.

21. ISO 9000 and ISO 14000 – in brief, ISO 9000. http://www.iso.org/iso/en/iso9000-14000/understand/inbrief.htm. Accessed August 16, 2006.

22. NTCIP 8007 – Testing and Conformity Assessment Documentation within NTCIP Standards Publications, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=8007. Accessed August 17, 2006.

23. NTCIP 1103 – Transportation Management Protocols, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=1103. Accessed August 17, 2006.

24. NTCIP 1204 – Environmental Sensor Station Interface Standard, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=1204. Accessed August 17, 2006.

25. NTCIP 1206 – Object Definitions for Data Collection and Monitoring (DCM) Device, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=1206. Accessed August 17, 2006.

26. NTCIP 1207 – Object Definitions for Ramp Meter Control (RMC) Units, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=1207. Accessed August 17, 2006.

27. NTCIP 1208 – Object Definitions for Closed Circuit Television (CCTV) Switching, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=1208. Accessed August 17, 2006.

28. NTCIP 1209 – Data Element Definitions for Transportation Sensor Systems, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=1209. Accessed August 17, 2006.

29. NTCIP 1210 – Field Management Stations - Part 1: Object Definitions for Signal System Masters, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=1210. Accessed August 17, 2006.

30. NTCIP 1211 – Object Definitions for Signal Control and Prioritization, A Joint Publication of AASHTO, ITE, and NEMA.

http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=1211. Accessed August 17, 2006.

31. NTCIP 1213 – Objects Definitions for Electrical and Lighting Management Systems, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=1213. Accessed August 17, 2006.

32. Tex-1170-T, Sampling and Environmental Testing of Traffic Signal Controller Assemblies: Traffic Signal Controllers and Conflict Monitors. Published by TxDOT. ftp://ftp.dot.state.tx.us/pub/txdot-info/cst/TMS/1100-T_series/pdfs/tsi1170.pdf. Accessed July 29, 2005.

33. Special Specifications 6504 – Testing, Training, Documentation and Warranty. Published by TxDOT. ftp://ftp.dot.state.tx.us/pub/txdot-info/cmd/cserve/specs/1993/spec/es6504.pdf. Accessed June 30, 2005.

34. NTCIP 9012 – Testing and Conformity Assessment User Guide for NTCIP Field Devices and Center-to-Field Communications, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=9012. Accessed August 31, 2006.

APPENDIX A
SPECIAL SPECIFICATION FOR CCTV EQUIPMENT

The following is a modified version of the TxDOT 2004 Special Specification 6025 – CCTV Field Equipment (*1*). The modification to the specification consists of adding a reference to another special specification defining the details of the NTCIP requirements, updating wording to be consistent with requirements in the NTCIP standards, and removing requirements that would be in conflict with or superseded by NTCIP requirements.

Italics highlight the additions to the document and strikethrough highlights the deleted wording.

SPECIAL SPECIFICATION
6025 *(Modified for NTCIP Requirements)*
CCTV Field Equipment

1. **Description.** Furnish and install closed circuit television (CCTV) field equipment.

   The following special specification is referenced in this specification: "National Transportation Communications for ITS Protocol for CCTV Equipment"

2. **Materials.** Provide new, corrosion resistant materials in accordance with the details shown on the plans and this item.

   Provide CCTV field equipment including, but not limited to, the following:

   - Color video camera units.
   - Camera lenses, filters, control circuits and accessories.
   - Camera housings.
   - Medium duty pan and tilt units.
   - Camera control receivers.
   - Video and camera control and power cable harnesses, connectors and coaxial cable.
   - Equipment for accommodating presets.
   - Source ID Generator.
   - When shown on the plans, Local Control Panel.

   A. **Functional Requirements.** Provide CCTV Cameras in accordance with NTSC and EIA-170A. Conform the system limiting resolution to FCC regulations for broadcast signals. Provide clear, low-bloom and low-lag video pictures under all conditions from bright sunlight to nighttime scene illumination of 0.1 ft.-candle (fc.). Maintain color quality by a continuous through the lens automatic white balance for color temperatures from 2850ºK to greater than 5100ºK with less than 10 IRE units unbalance.

Provide field equipment that operates in all weather conditions and able to withstand a wind load of 80 mph without permanent damage to mechanical and electrical equipment, unless otherwise shown on the plans.

Provide equipment from the same manufacturer at each field location.

B. Electrical and Mechanical Requirements.

1. **Video Camera Unit.** Provide color video cameras of solid state design, and that meet the following requirements:

- Use Digital Signal Processing (DSP):
  - For digital zoom;
  - For Auto/Manual long-term integration (exposure) control, with built-in frame buffer;
  - For Auto-focus;
  - For built-in I.D. Generator, with white letters and black outline.
- **Image Pickup Device:** 1/4 in. single chip interline transfer solid state color matrix CCD microlens sensor
- **Pickup Device Blemishes:** When viewing a uniform white field, there must be no blemishes for any iris opening producing any signal level between 7.5 and 100 IRE.
- **Sensitivity:** Maintain full p-p video with 0.1 fc. 3200°K incandescent illumination on the image device face plate with AGC off.
- **Resolution:** > 350 lines vertical and > 460 lines horizontal, measured per EIA-170A Standard.
- **Over Exposure Protection:** The camera must not sustain any permanent damage when pointed directly at strong light sources, including the sun, for brief periods of time.
- **Encoded NTSC Video Signal Format**: EIA-170A Standard, video output 1 Volt p-p composite. Must have up to 16 dB AGC.
- **Output Impedance:** 75 Ohms ± 5%.
- **Aspect Ratio**: 4:3.
- **Geometric Distortion**: Zero.
- **Signal to Noise Ratio (AGC Off):** 55 dB minimum (weighted at 4.5 MHz).
- Sensor with a minimum of 768(H) X 493(V) pixels.
- Lens must be integral to camera assembly.
- **Electronic Shutter Speed:** software selectable, remotely.

2. **Camera Lens.** Provide an integral lens assembly for each camera with the following features:

- An f/1.6 or better glass multi-coated zoom lens. The lens must have variable focal length from 3.9 mm to 85.8 mm.
- Provide motorized iris control with manual override with each lens.

Provide a lens with capabilities for remote control of zoom, focus and iris operations. Provide mechanical or electrical means to protect the motors from overrunning in extreme positions. The lens and controller system must be capable of both auto iris, and remote manual iris operation. Iris must be “motorized”, as opposed to “auto iris” type, for system control compatibility.

3. **Camera Housing.** Furnish and install an environmental resistant and tamperproof housing pressurized to 5 psi dry Nitrogen with Schrader purge fitting and 20 psi relief valve for each camera.

Except for the viewing window, construct the enclosure from 6061-T6 standard aluminum tubing with a wall thickness of 0.20 in. ± 0.03 in. Label internal wiring properly. Use a gas-tight connector at the rear plate of the housing.

The internal humidity of the housing must be less than 10%, when sealed and pressurized. Securely place desiccant packs inside the housing to absorb any residual moisture and maintain internal humidity at 10% or less.

Provide a low pressure sensor in the camera to put a “low-pressure” annotation on the video signal through the internal I.D. generator.

Construct the viewing window in such a way that unrestricted camera views can be obtained at all camera and lens positions.

Provide a sun shield to shield the entire housing from direct sunlight and vertical rainfall. Construct it in such a way as to allow the free passage of air between the housing and the shield, but it must not form a “sail” to place an excessive load on the pan/tilt unit in high winds.

Provide with an internal 15 W. low temperature heater with its own thermostat control in each housing.

Provide lightning protection as shown on the plans in each housing.

4. **Pan-Tilt Unit.** Furnish and install a medium duty, anodized aluminum weatherproof pan-and-tilt unit at each camera site on top of the camera pole. Provide a mounting plate to install the unit on the pole. Design the mounting for the camera housing and the pan-and-tilt unit to withstand the wind loading specified in Section 2.A.

Provide a unit with vertical movement of + 40° to – 90° and horizontal movement of 360° full, contiguous rotation movement. Tilt speed must be 20° per sec. and the pan speed must be up to 100° per sec. Provide a unit that is capable of simultaneous pan-and-tilt movements.

Provide a unit with a load rating compatible with that of the camera housing, camera and cabling under wind conditions specified in Section 2.A. and acceleration/deceleration conditions specified. Provide analyses of the loading on the pan-and-tilt assembly based on the above criteria.

Use Stepper motors.

Provide pan-and-tilt units that have seals and gaskets to protect the motors, gears, and cables. Provide seals and gaskets that are resistant to ozone, ultraviolet radiation, and other pollutants inherent to local environmental conditions.

5. **Local Control Panel.** Provide Local Control Panel that meet the following specific requirements without use of a laptop:

- Pan Left.
- Pan Right.
- Tilt Up.
- Tilt Down.
- Zoom In.
- Zoom Out.
- Focus Near.
- Focus Far.
- Manual and Auto Iris control.
- Iris Open.
- Iris Close.
- Pan/Tilt Position preset.
- Camera Power (Latching).
- Remote white balance control.
- Auto and Manual white balance control.
- Zoom and focus position preset.

6. **Control Receivers.** Mount the camera control receiver inside the camera unit. It must execute camera and lens functions and must also forward communication of commands for the pan/tilt functions to the pan/tilt control receiver. Mount the pan/tilt control receiver inside the pan/tilt unit. Provide camera and pan/tilt functions that are operable via RS-422 serial communications.

Provide control receivers that receive the command data from the camera controller and decode the digital command data signals transmitted through the communication transmission interface, perform error checking and act on valid data to drive the pan/tilt unit and the camera controls. Detail the communications transmission interface on the plans. Provide control receivers that are fully compatible with the existing camera controller shown on the plans.

Provide control receivers that meet the following specific requirements:

- **Camera remote control functions**: Provide units with, as a minimum, control and drive circuits for the following functions:
    - **DSP Functions**: Zoom, Long-Term Exposure, Auto-Focus, Auto/Manual focus Control, I.D. Generator Operation, and Alarm function Control.
    - Pan/Tilt Position preset.
    - Pan Left.
    - Pan Right.
    - Tilt Up.
    - Tilt Down.
    - Zoom and focus position preset.
    - Zoom In.
    - Zoom Out.
    - Focus Near.
    - Focus Far.
    - Manual and Auto Iris control.
        - Iris Open.
        - Iris Close.
    - Camera Power (Latching).
    - Remote white balance control.
    - Auto and Manual white balance control.
    - One auxiliary output (unless specified otherwise in the plans).
- **Controller Address**: Provide each unit with a unique programmable address. Provide units that respond to the central command if and only if they are addressed.
- **Power Supplies**: Provide power supplies required to operate the camera, pan/tilt, and lens movements and include them with the housing, camera control receiver, and pan/tilt unit.
- **Communications Interface:** Provide a camera control receiver that interfaces to the communications backbone through an *RS* ~~EIA-~~232 *Serial* ~~C/D~~ port *and shall be in accordance with special specification National Transportation Communications for ITS Protocol for CCTV Equipment*. When indicated on the plans, provide communications signals, data exchange protocol and timing that is compatible with the communications equipment and with the existing master controller in the satellite building. Use a minimum 9600 Baud data rate. Data must be sent asynchronously as ~~either~~ 8 bit with no parity.~~ or 7 bit with parity. Each block of data must include a camera identifier and be accompanied by a checksum calculated on the entire block. Blocks with a bad checksum must be NAKed. Block with a good checksum must be ACKed.~~ If the field unit must transmit data to the control unit at the Satellite Building,

it must raise the RTS line and keep it raised until all data has been sent. Provide a field unit that will not transmit data unless the CTS line from the communications equipment is raised. Provide the camera control receiver connectors and harness to connect to the communications equipment interface. ~~Supply complete hardware interface and protocol description to the Department as part of the required documentation.~~

Provide RS-232 to RS-422 external powered converter that is an integral part of the video communication junction box.

- **Power Input**: 115 VAC plus or minus 10%, 60 Hz ± 3 Hz, 50 W. Maximum.
- **Connectors**: Provide and install connectors which are compatible with the communications equipment interface. Use Connectors for connections at the pan/tilt mechanism. Make connections through a pigtail with a connector on it coming out of the bottom center of the pan/tilt unit. Provide the connector on the pigtail that is an AMP type connector. Provide connections down to the pole to the transmission cables to this connector. Supply mating connectors. Provide connector pins and mating connectors that are plated to ensure good electrical connection and resist corrosion. Use pressure tight multi-conductor MS-type cable connectors for camera connections.

7. **Source ID Generator.** Provide the built-in I.D. Generator that inserts camera ID over each of the camera generated videos.

Submit a list of proposed camera identification text to the Engineer for approval before the ID is programmed.

Once programmed, the programmed ID must automatically be displayed with its associated video signal.

Provide the source ID generator that will automatically “pass through” video in case of equipment failure.

When indicated on the plans, provide the source ID generator that is compatible with the existing camera controller shown on the plans.

8. **Video Communication Junction Box.** Install the video communication junction box in the CCTV equipment cabinet or in the surveillance cabinet, as shown on the plan and as directed by the Engineer. Provide the video communication junction box that contains the lightning protection devices for data, power, and video. The junction box must be grounded very well to the earth ground. Provide the junction box that has connectors for inputs and outputs for data, power, and video. Make testing and connections to communication devices through these external connectors.

9. **Surge Protection.** Provide the camera installation that meets the following requirements:

- Pole mounting adapter -- Electrically bonded to pole.
- Pan/tilt mechanism -- Electrically bonded to adapter.

- Camera housing -- Electrically bonded to pan/tilt unit.

10. **Power and Control Cable Surge Protector.** Protect each power conductor and each control conductor (including return conductors) by the appropriate surge protector. House the protective devices in each of the surveillance cabinets.

11. **Power Requirements.** Provide CCTV field equipment that meets its specified requirements when the input power is 115 VAC ± 10%, 60 Hz ± 3 Hz. The maximum power required must not exceed 350 W.

Provide equipment operations that are not affected by the transient voltages, surges and sags normally experienced on commercial power lines. Check the local power service to determine if any special design is needed for the equipment. The extra cost, if required, must be included in the bid of this item.

12. **Primary Input Power Interruption.** Provide CCTV field equipment that meets the requirements in Section 2.1.4. "Power Interruption" of the NEMA Standard TS2 for Traffic Control System.

13. **Power Service Transients.** Provide CCTV field equipment that meets the requirements of Section 2.1.6., "Transients, Power Service" of the NEMA Standard TS2.

14. **Wiring.** Provide wiring that meets the requirements of the National Electric Code. Provide wires that are cut to proper length before assembly. Do not doubled-back wire to take up slack. Lace wires neatly into cable with nylon lacing or plastic straps. Secure cables with clamps. Provide service loops at connections.

Provide coaxial cable between the camera and the communications equipment interface that is of the RG-59 type with a stranded center conductor and 100% shield coverage. Provide coaxial cable that has a cellular polyethylene dielectric.

15. **Transient Suppression.** Provide DC relays, solenoids and holding coils that have diodes or other protective devices across the coils for transient suppression.

16. **Power Service Protection.** Provide equipment that contains readily accessible, manually resettable or replaceable circuit protection devices (such as circuit breakers or fuses) for equipment and power source protection.

Provide and size circuit breakers or fuses such that no wire, component, connector, PC board or assembly must be subjected to sustained current in excess of their respective design limits upon the failure of any single circuit element or wiring.

17. **Fail Safe Provision.** Provide equipment that is designed such that the failures of the equipment must not cause the failure of any other unit of equipment.

18. **Modular Design.** Provide CCTV field equipment that is modular in design to allow major portions to be readily replaced in the field. Identify modules and

assemblies clearly with name, model number, serial number and any other pertinent information required to facilitate equipment maintenance.

19. **Connectors and Harnesses.** Provide external connections made by means of connectors. Provide connectors that are keyed to preclude improper hookups. Color code and/or appropriately mark wires to and from the connectors.

Provide connecting harnesses of appropriate length and terminated with matching connectors for interconnection with the communications system equipment.

Provide pins and mating connectors that are plated to improve conductivity and resist corrosion. Cover connectors utilizing solder type connections by a piece of heat shrink tubing securely shrunk to insure that it protects the connection.

C. Environmental Design Requirements. Provide equipment that meets its specified requirements during and after subjecting to any combination of the following conditions.

- Ambient temperature range of 0°F to 140°F.
- Temperature shock not to exceed 30°F per hour during which the relative humidity must not exceed 95%.
- Relative humidity range not to exceed 95% over the temperature range of 40°F to 110°F.
- Moisture condensation on exterior surfaces caused by temperature changes.

Provide camera and environmental housing assemblies that perform to stated specifications over an ambient temperature range of –35°F to +130°F and a humidity range of 0% to 100 % condensing. The camera must operate without sustaining damage over temperature range of –35°F to 140°F.

3. Construction Methods.

A. General. Provide equipment that utilizes the latest available techniques for design and construction with a minimum number of parts, subassemblies, circuits, cards, and modules to maximize standardization and commonality.

Design the equipment for ease of maintenance. Provide component parts that are readily accessible for inspection and maintenance. Provide test points that are for checking essential voltages and waveforms.

B. Electronic Components. Provide electronic components in accordance with Special Specification, “Electronic Components”.

C. Mechanical Components. Provide external screws, nuts and locking washers that are stainless steel; no self-tapping screws will be used. Provide parts made of corrosion resistant material, such as plastic, stainless steel, anodized aluminum or brass. Protect materials from fungus growth and moisture deterioration. Separate dissimilar metals by an inert dielectric material.

4. **Testing.** Perform testing in accordance with Article 2, Special Specification, "Testing, Training, Documentation, Final Acceptance, and Warranty".

5. **Training.** Provide training in accordance with Article 3, Special Specification, "Testing, Training, Documentation, Final Acceptance, and Warranty".

6. **Documentation.** Provide documentation in accordance with Article 4, Special Specification, "Testing, Training, Documentation, Final Acceptance, and Warranty".

7. **Warranty.** Provide a warranty in accordance with Article 6, Special Specification, "Testing, Training, Documentation, Final Acceptance, and Warranty".

8. **Measurement.** This Item will be measured as each unit furnished, installed, and tested.

9. **Payment.** The work performed and materials furnished in accordance with this Item and measured as provided under "Measurement" will be paid for at the unit price bid for "CCTV Field Equipment". This price is for equipment, cables and connectors; documentation and testing; and labor, materials, warranty, training and incidentals.

REFERENCES FOR APPENDIX A

1. Special Specifications 6025 – CCTV Field Equipment. Published by TxDOT. ftp://ftp.dot.state.tx.us/pub/txdot-info/cmd/cserve/specs/2004/spec/ss6025.pdf. Accessed June 21, 2006.

APPENDIX B
SPECIAL SPECIFICATION - NTCIP FOR CCTV EQUIPMENT

The following is a template for the wording for a new TxDOT 2004 Specification that defines NTCIP for CCTV equipment. The organization and content of this new specification is modeled after TxDOT Special Specification 6026 (*1*). Chapter 2 of this report provides an explanation of each item and the reasoning behind a number of choices. Also, please note the specification of the transport and subnetwork communications protocols are "direct connect" and based upon NTCIP 2101-PMPP/RS232 and NTCIP 2201-T2 (*2*,*3*). These protocols and standards reference nonnetworked serial communications. Networked communications protocols such as NTCIP 2104-Ethernet and NTCIP 2202-ITP are viable alternatives (*4*,*5*).

SPECIAL SPECIFICATION

XXXX

National Transportation Communications for ITS Protocol for CCTV Equipment

1. **Description.** Provide Closed Circuit Television (CCTV) software that complies with the National Transportation Communications for ITS Protocol (NTCIP).

2. **Requirements.** Ensure software complies with the NTCIP Standards when installed. Ensure software complies with the relevant current NTCIP standards, including associated amendments. The term "software" includes both software and firmware.

   [Official printed copies of the NTCIP Joint Standards Publications referenced in this specification may be purchased from Global Engineering Documents, phone 1-800-854-7179, or http://www.global.ihs.com. They are also freely available in Adobe Acrobat PDF format at http://www.ntcip.org/library/documents/.]

   A. Ensure software complies with NTCIP 2101 – Point-to-Multi-Point Protocol Using RS-232 Subnetwork Profile (2001) (direct connect).

   B. Ensure software complies with NTCIP 2201 – Transportation Transport Profile (v01.14), and shall meet the requirements of parsing method 1 and encapsulation method 1.

   C. Ensure software complies with NTCIP 2301 – Simple Transportation Management Framework – Application Profile (2001), and shall meet the requirements of Conformance Level 1 (SNMP).

D. Ensure software implements all mandatory objects of the mandatory and optional conformance groups as defined in NTCIP 1205 – Object Definitions for Closed Circuit Television (CCTV) Camera Control (v01.08a) as follows:

- CCTV Configuration with support for clauses 3.2, 3.3, and 3.11
- Extended Functions with support for clauses 3.6, 3.7, and 3.9
- Motion Control with support for clauses 3.4 and 3.5
- Configuration
- Security

E. Ensure software implements the following optional objects defined in NTCIP 1205 – Object Definitions for Closed Circuit Television (CCTV) Camera Control (v01.08a) as follows:

- positionQueryFocus
- positionQueryIris

F. Ensure software implements all mandatory objects of the mandatory conformance groups as defined in NTCIP 2301 – Simple Transportation Management Framework – Application Profile (2001):

- System Group
- SNMP Group
- SNMP Configuration

G. Ensure software implements the mandatory objects of the mandatory conformance groups as defined in NTCIP 2101 – Point-to-Multi-Point Protocol Using RS-232 Subnetwork Profile (2001):

- HDLC Group
- RS232 Asynchronous Group
- HDLC Group Address Group

H. Ensure that objects that are required to support this NTCIP requirement support all values within its standardized range. Standardization range is defined by a size, range, or enumerated listing indicated in the object's SYNTAX field and/or through the descriptive text in the object's description field of the relevant standard.

The following table provides the current listing of known variances for this project.

OBJECT	MINIMUM PROJECT REQUIREMENTS
NTCIP 1205	
zoneMaximum	16
labelMaximum	16
labelColor	7 and 16
rangeMaximumPreset	
rangePanLeftLimit	355 degrees (SS 6860) [1]
rangePanRightLimit	
rangePanHomePosition	
rangeTiltUpLimit	+90 degrees (SS 6287, SS 6860) +40 degrees (SS 6225) +20 degrees (SS 6973)
rangeTiltDownLimit	-90 degrees (SS 6287, SS 6860, SS 6225) -110 degrees (SS 6973)
rangeZoomLimit	
rangeFocusLimit	
rangeIrisLimit	
rangeMinimumPanStepAngle	
rangeMinimumTiltStepAngle	
systemCameraEquipped	
systemLensEquipped	
zoneCameraEquipped	
menuControl	1..9

[1] Researcher's Note: The SS XXXX numbers refer to TxDOT special specification numbers that define different values for the parameter.

Tilt Speed	3-4 degrees per second (SS 6287, SS 6142, SS 6860) 20 degrees per second (SS 6225) 7 degrees per second (SS 6973)
Pan Speed	5-6 degrees per second (SS 6287, SS 6142, SS 6860) 100 degrees per second (SS 6225) 10 degrees per second (SS 6973)

OBJECT	MINIMUM PROJECT REQUIREMENTS
NTCIP 1201	
communityNamesMax	3
maxGroupAddresses	1

I. Hardware Limitations: Ensure that a “noSuchName” SNMP error (OID not supported) will be returned by firmware for required objects that cannot be implemented due to hardware limitations.

J. Documentation: Ensure software is supplied with full documentation, including a CD-ROM containing ASCII versions of the following Management Information Base (MIB) files in Abstract Syntax Notation 1 (ASN.1) format:

- The relevant version of each official standard MIB Module referenced by the device functionality.
- A manufacturer-specific version of the official Standard MIB Module with the supported range indicated in ASN.1 format in the SYNTAX field of the associated OBJECT TYPE macro for devices that do not support the full range of any object within a Standard MIB Module. Ensure the file name is identical to the Standard MIB Module, except having the extension “.man”.
- A MIB containing any other objects supported by the device.

Allow unrestricted use of this documentation by any authorized party for systems integration purposes, regardless of what parties are involved in the systems integration effort.

Provide documentation of any procedural implementation details for function(s) shown in the following table, wherever multiple objects are required to implement a feature required by the CCTV Specification. The following table enumerates which objects are used and any special procedures or required sequences to implementing that feature of function. Submit this table for approval prior to implementation.

Function	Objects	Procedures to Implementation
Example: Function X	Object T Object Y Object Z	Get object T then send objects Y and Z if T>0.
Pan		
Tilt		
Zoom		
Focus		
Iris		
Presets		
Camera Feature Control		
Camera Zones and Labels		
Camera Timeouts		
Camera Range		
Camera Alarms		
NTCIP Security		
Retrieve Module Table		

3. **Testing and Verification.** Demonstrate conformance to the applicable sections of NTCIP using approved test software. Ensure that conformance testing is performed by a qualified independent testing firm and witnessed and certified by a Professional Engineer. Executable code and documentation for the test procedures shall be available upon request from:

   Mr. Carlos A. Lopez, P.E.
   Director, Traffic Operations Division
   Texas Department of Transportation
   125 E. 11th Street
   Austin, Texas 78701-2483

   The department reserves the right to have a representative witness all conformance tests. Test results will be compared to the requirements specified in this item. Failure to meet these requirements will be counted as a defect, and the software and associated hardware will be rejected. Final inspection and acceptance of software and associated hardware will be made after installation and performance testing at the designated locations as shown on the plans, unless otherwise directed.

4. **Measurement and Payment.** No direct measurement or payment will be made for the work performed and materials furnished in order to provide Closed Circuit Television (CCTV) software that complies with the National Transportation Communications for ITS Protocol (NTCIP) in accordance with this specification.

REFERENCES FOR APPENDIX B

1. Special Specifications 6026 – National Transportation Communications for ITS Protocol for Dynamic Message Signs. Published by TxDOT. ftp://ftp.dot.state.tx.us/pub/txdot-info/cmd/cserve/specs/2004/spec/ss6026.pdf. Accessed June 21, 2006.

2. NTCIP 2101– Point to Multi-Point Protocol Using RS-232 Subnetwork Profile, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=2101. Accessed June 27, 2006.

3. NTCIP 2201 – Transportation Transport Profile, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=2201. Accessed June 27, 2006.

4. NTCIP 2104 – Ethernet Subnetwork Profile, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=2104. Accessed July 25, 2005.

5. NTCIP 2202 – Internet (TCP/IP and UDP/IP) Transport Profile, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=2202. Accessed July 25, 2005.

APPENDIX C:
CCTV TEST PROCEDURES

INTRODUCTION

This appendix contains a set of test procedures for NTCIP conformant CCTV field equipment. Except for the prequalification test, the procedures are organized alphabetically by function and do not reflect any preferred sequence of execution.

TEST CASE SUMMARY

A summary of the test procedures and test cases for the features and/or functional areas derived from NTCIP 1205-CCTV is provided in Table C-1 (*1*). Two procedures, Global Configuration and Security, have a reference in NTCIP 1205-CCTV but the object definitions appear in NTCIP 1201-GLO (*1*,*2*). The majority of these test cases are derived from the Enterprise test procedures but are formatted to conform to NTCIP 8007-TEST (*3*,*4*). Some of the test procedures and/or test cases may not be applicable to CCTV field equipment that complies with TxDOT specifications because there is no TxDOT requirement for the feature or function. The test procedures do not address all the functional requirements of the TxDOT specification in that NTCIP does not support some of the TxDOT features or functions. Some test procedures also address features or functions that do not have a TxDOT specification requirement.

Table C-1. CCTV Test Case Summary.

CCTV Test Cases		
ID	**Title**	**Description**
Prequalification		
TC001	CCTV PRL Information	This procedure retrieves minimum project requirements and maximum values, checks for whether the required objects are implemented, and performs a sampling of the supported values.
Alarms		
TC001	Cabinet Alarm	Tests cabinet open alarm and label associated with it
TC002	Enclosure Alarm	Tests enclosure alarm and label **associated with it**
TC003	Video Loss Alarm	Tests video loss alarm and label **associated with it**
TC004	Temperature Alarm	Tests temperature alarm and label, thresholds, and current value **associated with it**
TC005	Pressure Alarm	Tests the pressure alarm and label, thresholds, and current value **associated with it**
TC006	Local Remote Alarm	Tests the local-remote alarm and label **associated with it**
TC007	Washer Fluid Alarm	Tests washer fluid alarm and label, thresholds, and current value **associated with it**
Configuration		
TC001	Identify Device	Tests whether device under test (DUT) contains valid information for the module make, model, and version number
TC002	Identify Preset Position Range	Ensures that device indicates that it supports the required number of preset positions.
TC003	Identify Pan Limits	Identifies and verifies the left and right panning limits and the home position of the device
TC004	True North Offset	Ensures that the user can configure the true north setting in the camera
TC005	Identify Tilt Limits	Ensures that the device indicates that it supports up and down tilting limits of the device
TC006	Identify Zoom Limits	Ensures that the device indicates that it supports the required zoom limit
TC007	Identify Focus Limits	Ensures that the device indicates that it supports the required focus limit

Table C-1. CCTV Test Case Summary (continued).

CCTV Test Cases		
ID	**Title**	**Description**
TC008	Identify Iris Limits	Ensures that the device indicates that it supports the required iris limit
TC009	Identify Pan-Tilt Step Angle Minimum	Ensures that the device indicates that it supports the pan and tilt step angle minimum
TC010	Identify Zone Functions	Ensures that the device indicates whether it supports zones, zone labels, and control within a zone
Discrete Input and Output		
TC001	Monitor Discrete Input	Verifies the state of discrete inputs and associated label
TC002	Monitor Discrete Output	Verifies the state of discrete outputs and associated label
Features		
TC001	Get Availability of Equipment	Identifies and verifies the availability of attached equipment to the camera
TC002	Control Camera Power	Enables and disables camera power while the user verifies
TC003	Control Heater Power	Enables and disables heater while user verifies
TC004	Control Wiper	Enables and disables wiper while user verifies
TC005	Control Washer	Enables and disables washer while user verifies
TC006	Control Blower	Enables and disables blower while user verifies
Focus		
TC001	Delta Focus Motion	Tests the delta focus motion of the camera by moving the camera with two different speeds and directions and allowing the user to verify them
TC002	Absolute Focus Motion	Tests the absolute focus motion of the camera by moving the camera with several different speed and direction parameters and allowing the user to verify them
TC003	Continuous Focus Motion with Timeout	Tests the continuous focus motion of the camera by moving the camera with the continuous command using the timeout parameter to stop the camera

Table C-1. CCTV Test Case Summary (continued).

CCTV Test Cases		
ID	**Title**	**Description**
TC004	Continuous Focus Motion with Stop	Tests the continuous focus motion of the camera by moving the camera and using the stop command to stop movement
Global Configuration		
TC001	Retrieve Module Table	This procedure retrieves the module table, and allows the tester to verify that the DUT reports the proper type of device, manufacturer, model, and version.
TC002	Global Set ID	This procedure ensures that a change to a static database object produces a change in globalSetIDParmeter.
Iris		
TC001	Delta Iris Motion	Tests the delta iris motion of the camera by moving the camera with the continuous command using the timeout parameter to stop the camera
TC002	Absolute Iris Motion	Tests the absolute iris motion of the camera by moving the camera with two different speeds and directions and allowing the user to verify them
TC003	Continuous Iris Motion with Timeout	Tests the continuous iris motion of the camera by moving the camera with the continuous command using the timeout parameter to stop the camera
TC004	Continuous Iris Motion with Stop	Tests the continuous iris motion of the camera by moving the camera and using the stop command to stop movement
Label		
TC001	Get and Set Label	Verifies the number of labels the device can store. Test labels are stored in the device to verify storage capabilities
TC002	Display Camera Location	Tests the capability of the device to display a text label on the video output

Table C-1. CCTV Test Case Summary (continued).

CCTV Test Cases		
ID	**Title**	**Description**
	Lens	
TC001	Get Availability of Lens Equipment	Identifies and verifies the availability of equipment attached to the camera
TC002	Control Auto Iris	Enables and disables the auto iris while the user verifies
TC003	Control Auto Focus	Enables and disables the auto focus while the user verifies
	Menu	
TC001	Menu	Tests the sending of menu commands to the CCTV while user verifies
	Pan	
TC001	Delta Pan Motion	Tests the delta panning motion of the camera by moving the camera with two different speeds and direction and allowing the user to verify them
TC002	Absolute Pan Motion	Tests the absolute panning motion of the camera by moving the camera with two different speeds and directions allowing the user to verify them
TC003	Continuous Pan Motion with Timeout	Tests the continuous panning motion of the camera by moving the camera with the continuous command using the timeout parameter to stop the camera
TC004	Continuous Pan Motion with Stop	Tests the continuous panning motion of the camera by moving the camera and using the stop command to stop movement
	Security	
TC001	Change Administrator Community Name	Verifies that the administrator can change the administrator community name stored in the DUT and properly affects operations
TC002	Change User Community Name	Verifies that the administrator can change the user community names and their masks stored in the DUT and properly affects operations

Table C-1. CCTV Test Case Summary (continued).

CCTV Test Cases		
ID	**Title**	**Description**
Tilt		
TC001	Delta Tilt Motion	Tests the delta tilt motion of the camera by moving the camera with two different speeds and directions and allowing the user to verify them
TC002	Absolute Tilt Motion	Tests the absolute tilt motion of the camera by moving the camera with two different speeds and directions and allowing the user to verify them
TC003	Continuous Tilt Motion with Timeout	Tests the continuous tilt motion of the camera by moving the camera twice with the continuous command using the timeout parameter to stop the camera
TC004	Continuous Tilt Motion with Stop	Tests the continuous tilting motion of the camera by moving the camera twice while using the stop command to stop movement
Zone		
TC001	Preset Position	Tests ability of the camera to store and move to preset camera positions
TC002	Get-Set Zone	Tests ability of the camera to store camera zones
TC003	Move In and Out of Zone	Tests the labeling capability of zones by moving to areas within zones
Zoom		
TC001	Delta Zoom Motion	Tests the delta zoom motion of the camera by moving the camera with two different speeds and directions and allowing the user to verify them
TC002	Absolute Zoom Motion	Tests the absolute zoom motion of the camera by moving the camera with two different speeds and directions and allowing the user to verify them
TC003	Continuous Zoom Motion with Timeout	Tests the continuous zoom motion of the camera by moving the camera twice with the continuous command using the timeout parameter to stop the camera

Table C-1. CCTV Test Case Summary (continued).

CCTV Test Cases		
ID	**Title**	**Description**
TC004	Continuous Zoom Motion with Stop	Tests the continuous zoom motion of the camera by moving the camera twice using the stop command to stop movement

TEST CASES

The details of each test case follow. The basic format of the test cases comes from the template that appears in NTCIP 8007-Test. As presented here the test case format has additional fields identifying constants and version history.

CCTV PRL Information

Test Case: **PRL-TC001**	*Title*: **CCTV PRL Information** Description: This procedure retrieves minimum project requirements and maximum values, checks for whether the required objects are implemented, and performs a sampling of the supported values for P and C objects. Constants: Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	CONFIGURE a list <CCTV Max and Static OIDs> that identifies read-only objects that either define maximum values that affect the indexes of tables or static variables that affect limits on other variables	
2.	CONFIGURE COMMUNITY NAME OUT = "administrator"	
3.	FOR ObjectName = each objectName in < CCTV Max and Static OIDs>	
4.	GET ObjectName	Pass/Fail
5.	Record the value on the PRL	
6.	NEXT ObjectName	
7.		
8.	CONFIGURE a list of objectNames that must be supported	
9.	FOR [objectName = each objectName in Supported objectName List	
10.	FOR all possible [instance values]	
11.	GET [objectName].instance *Note:* This loop performs the equivalent of a MIB Walk but uses GET instead of GET-NEXT.	Pass/Fail
12.	NEXT [instance value]	
13.	NEXT [objectName]	
14.	CONFIGURE a list <CCTV Test Values> that identifies instances of objects to test and a value in which to test the object with	
15.	FOR [objectNameInstance] = each objectNameInstance in <CCTV Test Values>	
16.	GET [objectNameInstance]	Pass/Fail
17.	RECORD RESPONSE VALUE in [currentValue]	
18.	FOR [testValue] = each objectNameValue in <CCTV Test Values>	
19.	SET [objectNameInstance] = [testValue]	Pass/Fail
20.	Record ObjectName, TestValue, and errorStatus	
21.	NEXT TestValue	
22.	SET [objectNameInstance] = [currentValue]	Pass/Fail
23.	NEXT ObjectName	
24.	RECORD responses on NTCIP PRL and note any anomalies	
	Test Case Results	
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 02/27/06 Implemented script and proofed – JJ	

Cabinet Alarm

Test Case: **Alarm-TC001**	*Title*: **Cabinet Alarm** Description: This Test Case tests the cabinet open alarm and label associated with it. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	SET labelText.<*alarmCLabIndex*> = <*alarmCLabText1*> labelHeight.<*alarmCLabIndex*> = <*alarmCLabHeight1*> labelColor.<*alarmCLabIndex*> = <*alarmCLabColor1*> labelStartRow.<*alarmCLabIndex*> = <*alarmCLabStartRow1*> labelStartColumn.<*alarmCLabIndex*> = <*alarmCLabStartColumn1*>	Pass/Fail
2.	SET alarmLabelIndex.0 to <*alarmCLabIndex*> 00 00 00 00 00 00	Pass/Fail
3.	USER VERIFY that no labels are being shown	Pass/Fail
4.	SET alarmLatchClear.0 to 0x00	Pass/Fail
5.	Turn on the alarm and USER VERIFY the label for the alarm is shown	Pass/Fail
6.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
7.	VERIFY RESPONSE VALUE alarmStatus = 0x80 alarmLatchStatus = 0x80	Pass/Fail
8.	SET alarmLatchClear.0 to 0x00	Pass/Fail
9.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
10.	VERIFY RESPONSE VALUE alarmStatus = 0x80 alarmLatchStatus = 0x80	Pass/Fail
11.	USER VERIFY the label for the alarm is shown and deactivate the alarm	Pass/Fail
12.	USER VERIFY the label for the alarm is off	Pass/Fail
13.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
14.	VERIFY Response Value alarmStatus = 0x00 alarmLatchStatus = 0x80	Pass/Fail
15.	USER VERIFY the label for the alarm is off	Pass/Fail
16.	SET alarmLatchClear.0 to 0x00	Pass/Fail
17.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
18.	VERIFY RESPONSE VALUE alarmStatus = 0x00 alarmLatchStatus = 0x00	Pass/Fail
Test Case Results		
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 11/03/05 Removed deprecated labelFontType – RDR V1.2 02/27/06 Implemented script and proofed – JJ	

Enclosure Alarm

Test Case: **Alarm-TC002**	*Title*: **Enclosure Alarm** Description: This Test Case tests the enclosure alarm and label associated with it. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	SET labelText.<*alarmELabIndex*> = <*alarmELabText1*> labelHeight.<*alarmELabIndex*> = <*alarmELabHeight1*> labelColor.<*alarmELabIndex*> = <*alarmELabColor1*> labelStartRow.<*alarmELabIndex*> = <*alarmELabStartRow1*> labelStartColumn.<*alarmELabIndex*> = <*alarmELabStartColumn1*>	Pass/Fail
2.	SET alarmLabelIndex.0 to 00 <*alarmELabIndex*> 00 00 00 00 00	Pass/Fail
3.	USER VERIFY that no labels are being shown	Pass/Fail
4.	SET alarmLatchClear.0 to 0x00	Pass/Fail
5.	Turn on the alarm and USER VERIFY the label for the alarm is shown	Pass/Fail
6.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
7.	VERIFY RESPONSE VALUE alarmStatus = 0x40 alarmLatchStatus = 0x40	Pass/Fail
8.	SET alarmLatchClear.0 to 0x00	Pass/Fail
9.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
10.	VERIFY RESPONSE VALUE alarmStatus = 0x40 alarmLatchStatus = 0x40	Pass/Fail
11.	USER VERIFY the label for the alarm is shown and deactivate the alarm	Pass/Fail
12.	USER VERIFY the label for the alarm is off	Pass/Fail
13.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
14.	VERIFY RESPONSE VALUE alarmStatus = 0x00 alarmLatchStatus = 0x40	Pass/Fail
15.	USER VERIFY the label for the alarm is off	Pass/Fail
16.	SET alarmLatchClear.0 to 0x00	Pass/Fail
17.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
18.	VERIFY RESPONSE VALUE alarmStatus = 0x00 alarmLatchStatus = 0x00	Pass/Fail
Test Case Results		
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 11/03/05 Removed deprecated labelFontType – RDR V1.2 02/27/06 Implemented script and proofed – JJ	

Video Loss Alarm

Test Case: **Alarm-TC003**	*Title*:	**Video Loss Alarm**
	Description:	This Test Case tests the video loss alarm and label associated with it.
	Variables:	
	Pass/Fail Criteria:	The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.

Test Step Number	*Test Procedure*	*Results*
1.	SET labelText.<*alarmVLabIndex*> = <*alarmVLabText1*> labelHeight.<*alarmVLabIndex*> = <*alarmVLabHeight1*> labelColor.<*alarmVLabIndex*> = <*alarmVLabColor1*> labelStartRow.<*alarmVLabIndex*> = <*alarmVLabStartRow1*> labelStartColumn.<*alarmVLabIndex*> = <*alarmVLabStartColumn1*>	Pass/Fail
2.	SET alarmLabelIndex.0 to 00 00 <*alarmVLabIndex*> 00 00 00 00	Pass/Fail
3.	USER VERIFY that no labels are being shown	Pass/Fail
4.	SET alarmLatchClear.0 to 0x00	Pass/Fail
5.	Turn on the alarm and USER VERIFY the label for the alarm is shown	Pass/Fail
6.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
7.	VERIFY RESPONSE VALUE alarmStatus = 0x20 alarmLatchStatus = 0x20	Pass/Fail
8.	SET alarmLatchClear.0 to 0x00	Pass/Fail
9.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
10.	VERIFY RESPONSE VALUE alarmStatus = 0x20 alarmLatchStatus = 0x20	Pass/Fail
11.	USER VERIFY the label for the alarm is shown and deactivate the alarm	Pass/Fail
12.	USER VERIFY the label for the alarm is off	Pass/Fail
13.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
14.	VERIFY RESPONSE VALUE alarmStatus = 0x00 alarmLatchStatus = 0x20	Pass/Fail
15.	USER VERIFY the label for the alarm is off	Pass/Fail
16.	SET alarmLatchClear.0 to 0x00	Pass/Fail
17.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
18.	VERIFY RESPONSE VALUE alarmStatus = 0x00 alarmLatchStatus = 0x00	Pass/Fail

Test Case Results				
Tested By:		*Date Tested:*		Pass/Fail
Test Case Notes:				
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 11/03/05 Removed deprecated labelFontType – RDR V1.2 02/27/06 Implemented script and proofed – JJ			

Temperature Alarm

Test Case: **Alarm-TC004**	*Title*: **Temperature Alarm** Description: This Test Case tests the temperature alarm and label, thresholds, and current value associated with it. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	SET labelText.<*alarmTLabIndex*> = <*alarmTLabText1*> labelHeight.<*alarmTLabIndex*> = <*alarmTLabHeight1*> labelColor.<*alarmTLabIndex*> = <*alarmTLabColor1*> labelStartRow.<*alarmTLabIndex*> = <*alarmTLabStartRow1*> labelStartColumn.<*alarmTLabIndex*> = <*alarmTLabStartColumn1*>	Pass/Fail
2.	SET alarmLabelIndex.0 to 00 00 00 <*alarmTLabIndex*> 00 00 00	Pass/Fail
3.	SET alarmTemperatureHighLowThreshold.0 to <*temperatureThreshold*>	Pass/Fail
4.	USER VERIFY that no labels are being shown	Pass/Fail
5.	SET alarmLatchClear.0 to 0x00	Pass/Fail
6.	Turn on the alarm and USER VERIFY the label for the alarm is shown	Pass/Fail
7.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
8.	VERIFY RESPONSE VALUE alarmStatus = 0x10 alarmLatchStatus = 0x10	Pass/Fail
9.	SET alarmLatchClear.0 to 0x00	Pass/Fail
10.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
11.	VERIFY RESPONSE VALUE alarmStatus = 0x10 alarmLatchStatus = 0x10	Pass/Fail
12.	USER VERIFY the label for the alarm is shown and deactivate the alarm	Pass/Fail
13.	USER VERIFY the label for the alarm is off	Pass/Fail
14.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
15.	VERIFY RESPONSE VALUE alarmStatus = 0x00 alarmLatchStatus = 0x10	Pass/Fail
16.	USER VERIFY the label for the alarm is off	Pass/Fail
17.	SET alarmLatchClear.0 to 0x00	Pass/Fail
18.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
19.	VERIFY RESPONSE VALUE alarmStatus = 0x00 alarmLatchStatus = 0x00	Pass/Fail
Test Case Results		
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 11/03/05 Removed deprecated labelFontType – RDR V1.2 02/27/06 Implemented script and proofed – JJ	

Pressure Alarm

Test Case: **Alarm-TC005**	*Title*: **Pressure Alarm** Description: This Test Case tests the pressure alarm and label, thresholds, and current value associated with it. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	SET labelText.<*alarmPLabIndex*> = <*alarmPLabText1*> labelHeight.<*alarmPLabIndex*> = <*alarmPLabHeight1*> labelColor.<*alarmPLabIndex*> = <*alarmPLabColor1*> labelStartRow.<*alarmPLabIndex*> = <*alarmPLabStartRow1*> labelStartColumn.<*alarmPLabIndex*> = <*alarmPLabStartColumn1*>	Pass/Fail
2.	SET alarmLabelIndex.0 to 00 00 00 00 <*alarmPLabIndex*> 00 00	Pass/Fail
3.	SET alarmPressureHighLowThreshold.0 to <*pressureThreshold*>	Pass/Fail
4.	USER VERIFY that no labels are being shown	Pass/Fail
5.	SET alarmLatchClear.0 to 0x00	Pass/Fail
6.	Turn on the alarm and USER VERIFY the label for the alarm is shown	Pass/Fail
7.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
8.	VERIFY RESPONSE VALUE alarmStatus = 0x08 alarmLatchStatus = 0x08	Pass/Fail
9.	SET alarmLatchClear.0 to 0x00	Pass/Fail
10.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
11.	VERIFY RESPONSE VALUE alarmStatus = 0x08 alarmLatchStatus = 0x08	Pass/Fail
12.	USER VERIFY the label for the alarm is shown and deactivate the alarm	Pass/Fail
13.	USER VERIFY the label for the alarm is off	Pass/Fail
14.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
15.	VERIFY RESPONSE VALUE alarmStatus = 0x00 and alarmLatchStatus = 0x08	Pass/Fail
16.	VERIFY the label for the alarm is off	Pass/Fail
17.	SET alarmLatchClear.0 to 0x00	Pass/Fail
18.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
19.	VERIFY RESPONSE VALUE alarmStatus = 0x00 alarmLatchStatus = 0x00	Pass/Fail
	Test Case Results	
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 11/03/05 Removed deprecated labelFontType – RDR V1.2 07/27/06 Implemented script and proofed – JJ	

Local Remote Alarm

Test Case: **Alarm-TC006**	*Title*: **Local Remote Alarm** Description: This Test Case tests the local-remote alarm and label associated with it. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	SET labelText.<*alarmLLabIndex*> = <*alarmLLabText1*> labelHeight.<*alarmLLabIndex*> = <*alarmLLabHeight1*> labelColor.<*alarmLLabIndex*> = <*alarmLLabColor1*> labelStartRow.<*alarmLLabIndex*> = <*alarmLLabStartRow1*> labelStartColumn.<*alarmLLabIndex*> = <*alarmLLabStartColumn1*>	Pass/Fail
2.	SET alarmLabelIndex.0 to 00 00 00 00 00 <*alarmLLabIndex*> 00	Pass/Fail
3.	USER VERIFY that no labels are being shown	Pass/Fail
4.	SET alarmLatchClear.0 to 0x00	Pass/Fail
5.	Turn on the alarm and USER VERIFY the label for the alarm is shown	Pass/Fail
6.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
7.	VERIFY RESPONSE VALUE alarmStatus = 0x04 alarmLatchStatus = 0x04	Pass/Fail
8.	SET alarmLatchClear.0 to 0x00	Pass/Fail
9.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
10.	VERIFY RESPONSE VALUE alarmStatus = 0x04 alarmLatchStatus = 0x04	Pass/Fail
11.	USER VERIFY the label for the alarm is shown and deactivate the alarm	Pass/Fail
12.	USER VERIFY the label for the alarm is off	Pass/Fail
13.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
14.	VERIFY RESPONSE VALUE alarmStatus = 0x00 alarmLatchStatus = 0x04	Pass/Fail
15.	USER VERIFY the label for the alarm is off	Pass/Fail
16.	SET alarmLatchClear.0 to 0x00	Pass/Fail
17.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
18.	VERIFY RESPONSE VALUE alarmStatus = 0x00 alarmLatchStatus = 0x00	Pass/Fail
	Test Case Results	
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 11/03/05 Removed deprecated labelFontType – RDR V1.2 02/27/06 Implemented script and proofed – JJ	

Washer Fluid Alarm

Test Case: **Alarm-TC007**	*Title*: **Washer Fluid Alarm** Description: This Test Case tests the washer fluid alarm and label, thresholds, and current value associated with it. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	SET labelText.<*alarmWFLabIndex*> = <*alarmWFLabText1*> labelHeight.<*alarmWFLabIndex*> = <*alarmWFLabHeight1*> labelColor.<*alarmWFLabIndex*> = <*alarmWFLabColor1*> labelStartRow.<*alarmWFLabIndex*> = <*alarmWFLabStartRow1*> labelStartColumn.<*alarmWFLabIndex*> = <*alarmWFLabStartColumn1*>	Pass/Fail
2.	SET alarmLabelIndex.0 to 00 00 00 00 00 00 <*alarmWFLabIndex*>	Pass/Fail
3.	SET alarmWasherFluidHighLowThreshold.0 to <washer*Threshold*>	Pass/Fail
4.	USER VERIFY that no labels are being shown	Pass/Fail
5.	SET alarmLatchClear.0 to 0x00	Pass/Fail
6.	Turn on the alarm and USER VERIFY the label for the alarm is shown	Pass/Fail
7.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
8.	VERIFY RESPONSE VALUE alarmStatus = 0x02 alarmLatchStatus = 0x02	Pass/Fail
9.	SET alarmLatchClear.0 to 0x00	Pass/Fail
10.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
11.	VERIFY RESPONSE VALUE alarmStatus = 0x02 alarmLatchStatus = 0x02	Pass/Fail
12.	USER VERIFY the label for the alarm is shown and deactivate the alarm	Pass/Fail
13.	USER VERIFY the label for the alarm is off	Pass/Fail
14.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
15.	VERIFY RESPONSE VALUE alarmStatus = 0x00 alarmLatchStatus = 0x02	Pass/Fail
16.	USER VERIFY the label for the alarm is off	Pass/Fail
17.	SET alarmLatchClear.0 to 0x00	Pass/Fail
18.	GET alarmStatus.0 and alarmLatchStatus.0	Pass/Fail
19.	VERIFY RESPONSE VALUE alarmStatus = 0x00 and alarmLatchStatus = 0x00	Pass/Fail
	Test Case Results	
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 11/03/05 Removed deprecated labelFontType – RDR V1.2 02/27/06 Implemented script and proofed – JJ	

Identify Device

Test Case: **Config-TC001**	*Title*: **Identify Device** Description: This Test Case ensures that the DUT contains valid information for the module make, model, and version number as well as other related information. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	GET globalMaxModules.0 = [globalMaxModules]	Pass/Fail
2.	VERIFY RESPONSE VALUE >= *<req_globalMaxModules>*	Pass/Fail
3.	FOR moduleIndex = 1 TO [globalMaxModules]	
4.	GET moduleDeviceNode.moduleIndex, moduleMake.moduleIndex, moduleModel.moduleIndex, moduleVersion.moduleIndex, and moduleType.moduleIndex.	Pass/Fail
5.	VERIFY moduleDeviceNode.moduleIndex returns 1.3.6.1.4.1.1206.4.2.7	Pass/Fail
6.	USER VERIFY RESPONSE VALUE moduleMake.moduleIndex = correct manufacturer name, moduleModel.moduleIndex = correct module number, moduleVersion.moduleIndex = correct version, and moduleType.moduleIndex = correct type of the tested device. *Note:* Make, Model, and Version are text descriptions. There should be at least 2 Types: hardware and software.	Pass/Fail
7.	NEXT moduleIndex	
Test Case Results		
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 02/22/06 Implemented script and proofed – JJ	

Identify Preset Position Range

Test Case: **Config-TC002**	*Title*: **Identify Preset Position Range** Description: This Test Case ensures that the device indicates that it supports the required number of preset positions. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	GET rangeMaximumPreset.0	Pass/Fail
2.	VERIFY RESPONSE VALUE >= *<req_rangeMaxPreset>*	Pass/Fail

Test Case Results				
Tested By:		*Date Tested:*		Pass/Fail
Test Case Notes:				
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 02/22/06 Implemented script and proofed – JJ			

Identify Pan Limits

Test Case: **Config-TC003**	*Title*: **Identify Pan Limits** Description: This Test Case identifies and verifies the left and right panning limits and the home position of the device. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	GET rangePanRightLimit.0 = [rangePanRightLimit] and rangePanLeftLimit.0 = [rangePanLeftLimit]	Pass/Fail
2.	VERIFY RESPONSE VALUE rangePanRightLimit >= <*req_rangePanRightLimit*>	Pass/Fail
3.	VERIFY RESPONSE VALUE rangePanLeftLimit <= <*req_rangePanLeftLimit*>	Pass/Fail
4.	SET positionPan.0 to Mode: 2 (Absolute), Speed: <*absolutePanSpeed*>, Position: 0, which is hex value 02 <*absolutePanSpeed*> 00 00	Pass/Fail
5.	SET positionTilt.0 to Mode: 2 (Absolute), Speed: <*absolutePanSpeed*>, Position: 0, which is hex value 02 <*absolutePanSpeed*> 00 00	Pass/Fail
6.	Make note current position of camera as Home position	
7.	SET positionPan.0 to Mode: 2 (Absolute), Speed: -<*absolutePanSpeed*>, Position: [rangePanLeftLimit], which is hex value 02 -<*absolutePanSpeed*> [rangePanLeftLimit]	Pass/Fail
8.	USER VERIFY camera panned to its left limit.	Pass/Fail
9.	SET positionPan.0 to Mode: 2 (Absolute), Speed: <*absolutePanSpeed*>, Position:0, which is hex value 02 <*absolutePanSpeed*> 00 00	Pass/Fail
10.	USER VERIFY camera moved back to its home position	Pass/Fail
11.	SET positionPan.0 to Mode: 2 (Absolute), Speed: <*absolutePanSpeed*>, Position:[rangePanRightLimit], which is hex value 02 <*absolutePanSpeed*> [rangePanRightLimit]	Pass/Fail
12.	USER VERIFY camera panned to its right limit	Pass/Fail

Test Case Results				
Tested By:		*Date Tested:*		Pass/Fail
Test Case Notes:				
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 02/22/06 Implemented script and proofed – JJ			

True North Offset

Test Case: **Config-TC004**	*Title*: **True North Offset** Description: This Test Case ensures that the user can configure the true north setting in the camera. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	GET rangeTrueNorthOffset.0 and record value to [rangeTrueNorthOffset]	Pass/Fail
2.	SET rangeTrueNorthOffset.0 to <*valid_rangeTrueNorthOffset*>	
3.	VERIFY RESPONSE ERROR = noError	Pass/Fail
4.	SET rangeTrueNorthOffset.0 to <*alternate_rangeTrueNorthOffset*>	
5.	VERIFY RESPONSE ERROR = badValue	Pass/Fail
6.	SET rangeTrueNorthOffset.0 to [rangeTrueNorthOffset]	Pass/Fail
	Test Case Results	
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 02/22/06 Implemented script and proofed – JJ	

Identify Tilt Limits

Test Case: **Config-TC0005**	*Title*: **Identify Tilt Limits** Description: This Test Case ensures that the device indicates that it supports up and down tilting limits of the device. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	GET rangeTiltDownLimit.0 and rangeTiltUpLimit.0	Pass/Fail
2.	VERIFY RESPONSE VALUE rangeTiltUpLimit >= <*req_rangeTiltUpLimit*>	Pass/Fail
3.	VERIFY RESPONSE VALUE rangeTiltDownLimit <= <*req_rangeTiltDownLimit*>	Pass/Fail
	Test Case Results	
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 02/22/06 Implemented script and proofed – JJ	

Identify Zoom Limits

Test Case: **Config-TC006**	*Title*: Description: Variables: Pass/Fail Criteria:	**Identify Zoom Limits** This Test Case ensures that the device indicates that it supports the required zoom limit. The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.		
Test Step Number	*Test Procedure*			*Results*
1.	GET rangeZoomLimit.0.			Pass/Fail
2.	VERIFY RESPONSE VALUE >= *<req_rangeZoomLimit>*			Pass/Fail
Test Case Results				
Tested By:		*Date Tested:*		Pass/Fail
Test Case Notes:				
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 02/22/06 Implemented script and proofed – JJ			

Identify Focus Limits

Test Case: **Config-TC007**	*Title*: Description: Variables: Pass/Fail Criteria:	**Identify Focus Limits** This Test Case ensures that the device indicates that it supports the required focus limit. The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.		
Test Step Number	*Test Procedure*			*Results*
1.	GET rangeFocusLimit.0			Pass/Fail
2.	VERIFY RESPONSE VALUE >= *<req_rangeFocusLimit>*			Pass/Fail
Test Case Results				
Tested By:		*Date Tested:*		Pass/Fail
Test Case Notes:				
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 02/22/06 Implemented script and proofed – JJ			

Identify Iris Limit

Test Case: **Config-TC008**	*Title*: Description: Variables: Pass/Fail Criteria:	**Identify Iris Limit** This test ensures that the device indicates that it supports the required iris limit. The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.		
Test Step Number	*Test Procedure*			*Results*
1.	GET rangeIrisLimit.0			Pass/Fail
2.	VERIFY RESPONSE VALUE >= *<req_rangeIrisLimit>*			Pass/Fail
Test Case Results				
Tested By:		*Date Tested:*		Pass/Fail
Test Case Notes:				
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 02/22/06 Implemented script and proofed – JJ			

Identify Pan-Tilt Step Angle Minimum

Test Case: **Config-TC009**	*Title*: Description: Variables: Pass/Fail Criteria:	**Identify Pan-Tilt Step Angle Minimum** This Test Case ensures that the device indicates that it supports the pan and tilt step angle minimum of the device. The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.		
Test Step Number	*Test Procedure*			*Results*
1.	GET rangeMinimumPanStepAngle.0 and rangeMinimumTiltStepAngle.0			Pass/Fail
2.	VERIFY RESPONSE VALUE rangeMinimumPanStepAngle <= *<req_rangeMinimumPanStepAngle>*			Pass/Fail
3.	VERIFY RESPONSE VALUE rangeMinimumTiltStepAngle <= *<req_rangeMinimumTiltStepAngle>*			Pass/Fail
Test Case Results				
Tested By:		*Date Tested:*		Pass/Fail
Test Case Notes:				
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 02/22/06 Implemented script and proofed – JJ			

Identify Zone Functions

Test Case: **Config-TC010**	*Title*:	**Identify Zone Functions**		
	Description:	This test ensures that the device indicates that it supports the required zone functions.		
	Variables:			
	Pass/Fail Criteria:	The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.		
Test Step Number	*Test Procedure*			*Results*
1.	GET zoneCameraEquipped.0			Pass/Fail
2.	VERIFY RESPONSE VALUE AND *<req_zoneCameraEquipped>* = *<req_zoneCameraEquipped>*			Pass/Fail
Test Case Results				
Tested By:		*Date Tested*		Pass/Fail
Test Case Notes:				
Version History:	V1.0 02/07/06 Initial Draft – RDR V1.1 02/22/06 Implemented script and proofed – JJ			

Monitor Discrete Input

Test Case: **Discrete-TC001**	*Title*:	**Monitor Discrete Input**	
	Description:	This Test Case verifies the state of discrete inputs and label associated with it.	
	Variables:		
	Pass/Fail Criteria:	The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*		*Results*
1.	SET labelText.*<inputLabelIndex1>* = *<inputLabText1>* labelHeight.*<inputLabelIndex1>* = *<inputLabHeight1>* labelColor.*<inputLabelIndex1>* = *<inputLabColor1>* labelStartRow.*<inputLabelIndex1>* = *<inputLabStartRow1>* labelStartColumn.*<inputLabelIndex1>* = *<inputLabStartColumn1>*		Pass/Fail
2.	SET byte *<input1>* of inputLabelIndex.0 to *<inputLabelIndex1>*		Pass/Fail
3.	Turn off inputs and USER VERIFY no labels for the corresponding inputs are shown		Pass/Fail
4.	SET inputLatchClear.0 to 0x00		Pass/Fail
5.	Turn on the input and USER VERIFY the label for the input is shown		Pass/Fail
6.	GET inputStatus.0 and inputLatchStatus.0		Pass/Fail
7.	VERIFY RESPONSE VALUE inputStatus & 2^(*<input1>*-1) = inputStatus inputLatchStatus & 2^(*<input1>*-1) = inputLatchStatus *Note:* This test will verify that bits *<input1>* are on.		Pass/Fail
8.	USER VERIFY the label for the corresponding input is on		Pass/Fail
9.	SET inputLatchClear.0 to 0x00		Pass/Fail
10.	GET inputStatus.0 and inputLatchStatus.0		Pass/Fail

11.	VERIFY RESPONSE VALUE inputStatus & 2^(<*input1*>-1) = inputStatus inputLatchStatus & 2^(<*input1*>-1) = inputLatchStatus *Note:* This test will verify that bits <*input*> are on.	Pass/Fail
12.	USER VERIFY the label for the input is shown and deactivate the input	Pass/Fail
13.	GET inputStatus.0 and inputLatchStatus.0	Pass/Fail
14.	VERIFY RESPONSE VALUE inputStatus & 2^(<*input1*>-1) = 0 inputLatchStatus & 2^(<*input1*>-1) = inputLatchStatus *Note:* This test will verify that bits <*input1*> are on (1) or off (0).	Pass/Fail
15.	USER VERIFY the label for the input is off	Pass/Fail
16.	SET inputLatchClear.0 to 0x00.	Pass/Fail
17.	GET inputStatus.0 and inputLatchStatus.0	Pass/Fail
18.	VERIFY RESPONSE VALUE inputStatus & 2^(<*input1*>-1) = 0 inputLatchStatus & 2^(<*input1*>-1) = 0 *Note:* This test will verify that bits <*input1*> are off.	Pass/Fail
19.	USER VERIFY the label for the input is off	Pass/Fail
	Test Case Results	
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 11/03/05 Removed deprecated labelFontType Corrected comparison values for inputStatus and inputLatchStatus – RDR V1.2 02/27/06 Implemented script and proofed – JJ	

Monitor Discrete Output

Test Case: **Discrete-TC002**	*Title*: **Monitor Discrete Output** Description: This Test Case verifies the state of discrete outputs and label associated with it. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	GET labelText.<*outputLabelIndex1*> = [labelText1] labelHeight.<*outputLabelIndex1*> = [labelHeight1] labelColor.<*outputLabelIndex*1> = [labelColor1] labelStartRow.<*outputLabelIndex1*> = [labelStartRow1] labelStartColumn.<*outputLabelIndex1*> = [labelStartColumn1]	Pass/Fail
2.	GET labelText.<*outputLabelIndex2*> = [labelText2] labelHeight.<*outputLabelIndex2*> = [labelHeight2] labelColor.<*outputLabelIndex*2> = [labelColor2] labelStartRow.<*outputLabelIndex2*> = [labelStartRow2] labelStartColumn.<*outputLabelIndex2*> = [labelStartColumn2]	Pass/Fail
3.	SET labelText.<*outputLabelIndex1*> = <*outputLabText1*> labelHeight.<*outputLabelIndex1*> = <*outputLabHeight1*> labelColor.<*outputLabelIndex1*> = <*outputLabColor1*> labelStartRow.<*outputLabelIndex1*> = <*outputLabStartRow1*> labelStartColumn.<*outputLabelIndex1*> = <*outputLabStartColumn1*>	Pass/Fail
4.	SET labelText.<*outputLabelIndex2*> = <*outputLabText2*> labelHeight.<*outputLabelIndex2*> = <*outputLabHeight2*> labelColor.<*outputLabelIndex2*> = <*outputLabColor2*> labelStartRow.<*outputLabelIndex2*> = <*outputLabStartRow2*> labelStartColumn.<*outputLabelIndex2*> = <*outputLabStartColumn2*>	Pass/Fail
5.	SET outputControl.0 to 0x0000	Pass/Fail
6.	SET byte <*output1*> and <*output2*> of outputLabelIndex.0 to <*outputLabelIndex1*> and <*outputLabelIndex2*>	Pass/Fail
7.	SET outputControl.0 to 0xXX 0x10 *Note:* The first byte, 0xXX, should be the value where only the bit for <*output1*> is on. For example 0x04 for output1 = 3.	Pass/Fail
8.	USER VERIFY that <*output1*> is on	Pass/Fail
9.	GET outputStatus.0	Pass/Fail
10.	VERIFY RESPONSE VALUE AND (2^<*output1*>-1) = (2^<*output1*>-1) *Note:* This test will verify that bit <*output1*> is on.	Pass/Fail
11.	SET outputControl.0 to 0xXX 0x10 *Note:* The first byte, 0xXX, should be the value where only the bit	Pass/Fail

	for <*output2*> is on	
12.	USER VERIFY that <*output2*> is on	Pass/Fail
13.	GET outputStatus.0	Pass/Fail
14.	VERIFY RESPONSE VALUE (2^<*output2*>-1) = (2^<*output2*>-1). *Note:* This test will verify that bit <*output2*> is on.	Pass/Fail
15.	SET outputControl.0 to 0x0000	Pass/Fail
16.	USER VERIFY that all outputs are off	Pass/Fail
17.	GET outputStatus.0	Pass/Fail
18.	VERIFY RESPONSE VALUE AND (2^<*output1*>-1) + (2^<*output2*>-1) = 0	Pass/Fail
19.	SET labelText.<*outputLabelIndex1*> = [labelText1] labelHeight.<*outputLabelIndex1*> = [labelHeight1] labelColor.<*outputLabelIndex1*> = [labelColor1] labelStartRow.<*outputLabelIndex1*> = [labelStartRow1] labelStartColumn.<*outputLabelIndex1*> = [labelStartColumn1]	Pass/Fail
20.	SET labelText.<*outputLabelIndex2*> = [labelText2] labelHeight.<*outputLabelIndex2*> = [labelHeight2] labelColor.<*outputLabelIndex2*> = [labelColor2] labelStartRow.<*outputLabelIndex2*> = [labelStartRow2] labelStartColumn.<*outputLabelIndex2*> = [labelStartColumn2]	Pass/Fail
	Test Case Results	
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 11/03/05 Removed deprecated labelFontType – RDR V1.2 02/27/06 Implemented script and proofed – JJ	

Get Availability of Equipment

Test Case: **Features-TC001**	*Title*: **Get Availability of Equipment** Description: This Test Case identifies and verifies the availability of attached equipment to the camera. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.			
Test Step Number	*Test Procedure*			*Results*
1.	User Enter the available equipment and store it to <*req_systemCameraEquipped*>			
2.	GET systemCameraEquipped.0			Pass/Fail
3.	VERIFY RESPONSE VALUE = <*req_systemCameraEquipped*>			Pass/Fail
Test Case Results				
Tested By:		*Date Tested:*		Pass/Fail
Test Case Notes:				
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 02/27/06 Implemented script and proofed – JJ			

Control Camera Power

Test Case: **Features-TC002**	*Title*: **Control Camera Power** Description: This Test Case enables and disables this feature while the user verifies. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.			
Test Step Number	*Test Procedure*			*Results*
1.	GET systemCameraEquipped.0			Pass/Fail
2.	VERIFY that the camera supports controlling camera power			Pass/Fail
3.	GET systemCameraFeatureControl.0			Pass/Fail
4.	SET systemCameraFeatureControl.0 to 0x8080			Pass/Fail
5.	DELAY 3 seconds			
6.	GET systemCameraFeatureStatus.0			Pass/Fail
7.	VERIFY that bit 7 is on and camera power is ON			Pass/Fail
8.	SET systemCameraFeatureControl.0 to 0x8000			Pass/Fail
9.	DELAY 3 seconds			
10.	GET systemCameraFeatureStatus.0			Pass/Fail
11.	VERIFY that bit 7 is off and camera power is OFF			Pass/Fail
12.	SET systemCameraFeatureControl.0 = 0x8080			Pass/Fail
Test Case Results				
Tested By:		*Date Tested:*		Pass/Fail
Test Case Notes:				
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 11/03/05 Corrected systemCameraFeatureControl settings – RDR V1.2 01/31/06 Added Delay to test – RDR V1.3 02/27/06 Implemented script and proofed – JJ			

Control Heater Power

Test Case: **Features-TC003**	*Title*:	**Control Heater Power**	
	Description:	This Test Case enables and disables this feature while the user verifies.	
	Variables:		
	Pass/Fail Criteria:	The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*		*Results*
1.	GET systemCameraEquipped.0		Pass/Fail
2.	VERIFY that the camera supports controlling heater power		Pass/Fail
3.	GET systemCameraFeatureControl.0 = [systemFeatureControl]		Pass/Fail
4.	SET systemCameraFeatureControl.0 to 0x40800		Pass/Fail
5.	GET systemCameraFeatureStatus.0		Pass/Fail
6.	VERIFY that bit 6 is on and heater power is ON		Pass/Fail
7.	SET systemCameraFeatureControl.0 to 0x4000		Pass/Fail
8.	GET systemCameraFeatureStatus.0		Pass/Fail
9.	VERIFY that bit 6 is off and heater power is OFF		Pass/Fail
10.	SET systemCameraFeatureControl.0 = [systemFeatureControl]		Pass/Fail
Test Case Results			
Tested By:		*Date Tested:*	Pass/Fail
Test Case Notes:			
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 11/03/05 Corrected systemCameraFeatureControl settings – RDR V1.2 02/27/06 Implemented script and proofed – JJ		

Control Wiper

Test Case: **Features-TC004**	*Title*:	**Control Wiper**	
	Description:	This Test Case enables and disables this feature while the user verifies.	
	Variables:		
	Pass/Fail Criteria:	The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*		*Results*
1.	GET systemCameraEquipped.0		Pass/Fail
2.	VERIFY that the camera supports controlling wiper		Pass/Fail
3.	GET systemCameraFeatureControl.0 = [systemFeatureControl]		Pass/Fail
4.	SET systemCameraFeatureControl.0 to 0x2080		Pass/Fail
5.	GET systemCameraFeatureStatus.0		Pass/Fail
6.	VERIFY that bit 5 is on and wiper is ON		Pass/Fail
7.	SET systemCameraFeatureControl.0 to 0x2000		Pass/Fail
8.	GET systemCameraFeatureStatus.0		Pass/Fail
9.	VERIFY that bit 5 is off and wiper is OFF		Pass/Fail
10.	SET systemCameraFeatureControl.0 = [systemFeatureControl]		Pass/Fail
Test Case Results			
Tested By:		*Date Tested:*	Pass/Fail

Test Case Notes:	
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 11/03/05 Corrected systemCameraFeatureControl settings – RDR V1.2 02/27/06 Implemented script and proofed – JJ

Control Washer

Test Case: **Features-TC005**	*Title*: Description: Variables: Pass/Fail Criteria:	**Control Washer** This Test Case enables and disables this feature while the user verifies. The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.		
Test Step Number	*Test Procedure*			*Results*
1.	GET systemCameraEquipped.0 = [systemCameraEquipped]			Pass/Fail
2.	VERIFY that the camera supports controlling washer			Pass/Fail
3.	GET systemCameraFeatureControl.0			Pass/Fail
4.	SET systemCameraFeatureControl.0 to 0x1080			Pass/Fail
5.	GET systemCameraFeatureStatus.0			Pass/Fail
6.	VERIFY that bit 4 is on and washer is ON			Pass/Fail
7.	SET systemCameraFeatureControl.0 to 0x1000			Pass/Fail
8.	GET systemCameraFeatureStatus.0			Pass/Fail
9.	VERIFY that bit 4 is off and washer is OFF			Pass/Fail
10.	SET systemCameraFeatureControl.0 = [systemCameraEquipped]			Pass/Fail
Test Case Results				
Tested By:		*Date Tested:*		Pass/Fail
Test Case Notes:				
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 11/03/05 Corrected systemCameraFeatureControl settings – RDR V1.2 02/27/06 Implemented script and proofed – JJ			

Control Blower

Test Case: **Features-TC006**	*Title*: Description: Variables: Pass/Fail Criteria:	**Control Blower** This Test Case enables and disables this feature while the user verifies. The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*		*Results*
1.	GET systemCameraEquipped.0		Pass/Fail
2.	VERIFY that the camera supports controlling blower		Pass/Fail
3.	GET systemCameraFeatureControl.0 = [systemFeatureControl]		Pass/Fail
4.	SET systemCameraFeatureControl.0 to 0x0880		Pass/Fail
5.	GET systemCameraFeatureStatus.0		Pass/Fail
6.	VERIFY that bit 3 is on and blower is ON		Pass/Fail
7.	SET systemCameraFeatureControl.0 to 0x0800		Pass/Fail

8.	GET systemCameraFeatureStatus.0	Pass/Fail
9.	VERIFY that bit 3 is off and blower is OFF	Pass/Fail
10.	SET systemCameraFeatureControl.0 = [systemFeatureControl]	Pass/Fail
Test Case Results		
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 11/03/05 Corrected systemCameraFeatureControl settings – RDR V1.2 02/27/06 Implemented script and proofed – JJ	

Delta Focus Motion

Test Case: **Focus-TC001**	*Title*: **Delta Focus Motion** Description: This Test Case tests the delta focus motion of the camera by moving the camera with several different speed and direction parameters and allowing the user to verify them. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	GET rangeFocusLimit.0	Pass/Fail
2.	VERIFY camera supports focus limits	Pass/Fail
3.	GET systemLensFeatureControl.0 = [systemLensFeatureControl]	Pass/Fail
4.	SET systemLensFeatureControl.0 to 0x4000	Pass/Fail
5.	SET positionFocusLens.0 to 01 <*deltaFocusMoveSpeed*> <*deltaFocusMovement*>	Pass/Fail
6.	USER VERIFY the camera lens moved towards far focus at the movement and speed specified by the test variables *<deltaFocusMoveSpeed> and <deltaFocusMovement>*	Pass/Fail
7.	SET positionFocusLens.0 to 01 -<*deltaFocusMoveSpeed*> <*deltaFocusMovement*>	Pass/Fail
8.	USER VERIFY the camera lens moved towards near focus at the movement and speed specified by the test variables *<deltaFocusMoveSpeed> and <deltaFocusMovement>*	Pass/Fail
9.	SET systemLensFeatureControl.0 = [systemLensFeatureControl]	Pass/Fail
Test Case Results		
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 02/06/06 Added step to turn auto focus on/off for test – JJ V1.2 02/13/06 Added test for support of focus limits – JJ V1.3 02/27/06 Implemented script and proofed – JJ	

Absolute Focus Motion

Test Case: **Focus-TC002**	*Title*: **Absolute Focus Motion** Description: This Test Case tests the absolute focus motion of the camera by moving the camera with several different speed and direction parameters and allowing the user to verify them. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	GET rangeFocusLimit.0	Pass/Fail
2.	VERIFY camera supports focus limits	Pass/Fail
3.	SET positionFocusLens.0 to 02 00 00	Pass/Fail
4.	GET systemLensFeatureControl = [systemLensFeatureControl]	Pass/Fail
5.	SET systemLensFeatureControl.0 to 0x4000	Pass/Fail
6.	SET positionFocusLens.0 to 02 <*absoluteFocusSpeed*> <*absoluteFocusPosition*>	Pass/Fail
7.	USER VERIFY the camera moved to the position defined by <*absoluteFocusPosition*>	Pass/Fail
8.	GET positionQueryFocus.0	Pass/Fail
9.	VERIFY RESPONSE VALUE = <*absoluteFocusPosition*>	Pass/Fail
10.	SET positionFocusLens.0 to 02 <*absoluteFocusSpeed*> <*absoluteFocusPosition2*>	Pass/Fail
11.	USER VERIFY the camera moved to the position defined by <*absoluteFocusPosition2*>	Pass/Fail
12.	GET positionQueryFocus.0	Pass/Fail
13.	VERIFY RESPONSE VALUE = <*absoluteFocusPosition2*>	Pass/Fail
14.	SET systemLensFeatureControl.0 = [systemLensFeatureControl]	Pass/Fail
Test Case Results		
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 11/03/05 Added test for positionQueryFocus – JJ V1.2 02/06/06 Added step to turn auto focus on/off for test – RDR V1.3 02/13/06 Added test for support of focus limits Added step to set position to Home position to test absolute movements – JJ V1.4 02/27/06 Implemented script and proofed – JJ	

Continuous Focus Motion with Timeout

Test Case: **Focus-TC003**	*Title*: **Continuous Focus Motion with Timeout** Description: This Test Case tests the continuous focus motion of the camera by moving the camera with the continuous command using the timeout parameter to stop the camera. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	GET systemLensFeatureControl = [systemLensFeatureControl]	Pass/Fail
2.	SET systemLensFeatureControl.0 to 0x4000	Pass/Fail
3.	GET timeoutFocus.0 = [timeoutFocus]	Pass/Fail
4.	SET timeoutFocus.0 to <*alt_contFocusTimeout*>	Pass/Fail
5.	SET positionFocusLens.0 to 03 <*conFocusSpeed*> 00 00	Pass/Fail
6.	USER VERIFY the camera lens stops moving in a far focus direction after <*alt_contFocusTimeout*> milliseconds	Pass/Fail
7.	SET positionFocusLens.0 to 03 -<*conFocusSpeed*> 00 00	Pass/Fail
8.	USER VERIFY the camera lens stops moving in a near focus direction after <*alt_contFocusTimeout*> milliseconds	Pass/Fail
9.	SET timeoutFocus.0 back = [timeoutFocus]	Pass/Fail
10.	SET systemLensFeatureControl.0 = [systemLensFeatureControl]	Pass/Fail
Test Case Results		
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 02/06/06 Added step to turn auto focus on/off for test – JJ V1.2 02/27/06 Implemented script and proofed – JJ	

Continuous Focus Motion with Stop

Test Case: **Focus-TC004**	*Title*: **Continuous Focus Motion with Stop** Description: This Test Case tests the continuous focus motion of the camera by moving the camera and using the stop command to stop movement. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	GET systemLensFeatureControl = [systemLensFeatureControl]	Pass/Fail
2.	SET systemLensFeatureControl.0 to 0x4000	Pass/Fail
3.	GET timeoutFocus.0 = [timeoutFocus]	Pass/Fail
4.	SET timeoutFocus.0 to 0	Pass/Fail
5.	SET positionFocusLens.0 to 03 <*conFocusSpeed*> 00 00	Pass/Fail
6.	DELAY <*alt_contFocusTimeout*> milliseconds	
7.	SET positionFocusLens.0 to 00 00 00 00	Pass/Fail
8.	USER VERIFY the camera stops moving	Pass/Fail
9.	SET positionFocusLens.0 to 03 -<*conFocusSpeed*> 00 00	Pass/Fail

10.	DELAY <*alt_contFocusTimeout*> milliseconds.	
11.	SET positionFocusLens.0 to 00 00 00 00	Pass/Fail
12.	USER VERIFY the camera stops moving	Pass/Fail
13.	SET timeoutFocus.0 back = [timeoutFocus]	Pass/Fail
14.	SET systemLensFeatureControl.0 = [systemLensFeatureControl]	Pass/Fail
Test Case Results		
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 02/06/06 Added step to turn auto focus on/off for test – JJ V1.2 02/27/06 Implemented script and proofed – JJ	

Retrieve Module Table

Test Case: **GloCon-TC001**	*Title*: **Retrieve Module Table** Description: This Test Case retrieves the module table, and allows the tester to verify that the DUT reports the proper type of device, manufacturer, model, and version. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	GET globalMaxModules.0 = [globalMaxModules]	Pass/Fail
2.	FOR N = 1 TO [globalMaxModules]	
3.	GET the following objects: moduleNumber.N, moduleDeviceNode.N, moduleMake.N, moduleModel.N, moduleVersion.N, and moduleType.N	Pass/Fail
4.	VERIFY RESPONSE VALUE for moduleNumber.N = N	Pass/Fail
5.	VERIFY RESPONSE VALUE for moduleDeviceNode.N = <OID for device type> *Note:* Per NTCIP 8004 v01.37, the following are <OID for device type> values: signal controller = 1.3.6.1.4.1.1206.4.2.1 ramp controller = 1.3.6.1.4.1.1206.4.2.2 dms = 1.3.6.1.4.1.1206.4.2.3 tss = 1.3.6.1.4.1.1206.4.2.4 ess = 1.3.6.1.4.1.1206.4.2.5 cctv = 1.3.6.1.4.1.1206.4.2.7 cctvSwitch = 1.3.6.1.4.1.1206.4.2.8 dcm = 1.3.6.1.4.1.1206.4.2.9 ssm = 1.3.6.1.4.1.1206.4.2.10 scp = 1.3.6.1.4.1.1206.4.2.11 network Camera = 1.3.6.1.4.1.1206.4.2.12 elms = 1.3.6.1.4.1.1206.4.2.13	Pass/Fail

6.	USER VERIFY RESPONSE VALUE for moduleMake.N indicates the manufacturer of the module *Note:* This might be the manufacturer of the hardware if the moduleType.N is hardware, or the developer of the software, if the moduleType.N is software.	Pass/Fail
7.	USER VERIFY RESPONSE VALUE for moduleModel.N indicates the correct model number of the component	Pass/Fail
8.	USER VERIFY RESPONSE VALUE for moduleVersion.N indicates the correct version number for the component	Pass/Fail
9.	USER VERIFY RESPONSE VALUE for moduleType.N indicates the correct type *Note:* Values correspond to the following: other (1) hardware (2) software (3)	Pass/Fail
10.	NEXT	
Test Case Results		
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 Original as defined in NTCIP 8007 v01.20 - B.3.2 Retrieve Module Table V2.0 02/22/06 Reformatted to use FOR and NEXT and simplified a number of steps. Changed “Change” to CONFIGURE Changed OID for device type to <OID for device type> Removed “Temporary community name” from Variables field and adopted convention of using [name of variable] for the text description of local variables and using <name> for externally defined constants Added list of moduleDeviceNodes from NTCIP 8004 – RDR V2.1 02/22/06 Implemented script and proofed – JJ	

Global Set ID

Test Case: **GloCon-TC002**	*Title*: **Global Set ID** Description: This procedure ensures that a change in a static database object produces a change in globalSetIDParmeter. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	GET globalSetIDParmeter.0 = [globalSetIDParameter]	Pass/Fail
2.	GET <*setOctStringParameterOID*> = [octStringOriginalValue] *Note:* The <*setOctStringParameterOID*> can be any object with SYNTAX of OCTET STRING that is also is static database object.	Pass/Fail
3.	SET <*setOctStringParameterOID*> = <setOctStringParameterValue>	Pass/Fail
4.	DELAY 120 Seconds	

	Note: The exact frequency for updating the globalSetIDParmeter.0 is manufacturer specific. The delay may have to be adjusted accordingly.	
5.	GET globalSetIDParmeter.0	Pass/Fail
6.	VERIFY RESPONSE VALUE ≠ [globalSetIDParameter]	Pass/Fail
7.	SET <*setOctStringParameterOID*> = [octStringOriginalValue]	Pass/Fail
8.	GET globalSetIDParmeter.0 = [globalSetIDParameter]	Pass/Fail
9.	GET <*setIntegerParameterOID*> = [integerOriginalValue] *Note:* The <*setIntegerParameterOID*> can be any object with SYNTAX of Integer that is also is static database object.	Pass/Fail
10.	SET <*setIntegerParameterOID*> = <*setIntegerParameterValue*>	Pass/Fail
11.	DELAY 120 seconds *Note:* The exact frequency for updating the globalSetIDParmeter.0 is manufacturer specific. The delay may have to be adjusted accordingly.	
12.	GET globalSetIDParmeter.0	Pass/Fail
13.	VERIFY RESPONSE VALUE ≠ [globalSetIDParameter]	Pass/Fail
14.	SET <*setIntegerParameterOID*> = [integerOriginalValue]	Pass/Fail
	Test Case Results	
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 02/10/06 Initial procedure defined – RDR V1.1 02/22/06 Implemented script and proofed – JJ	

Delta Iris Motion

Test Case: **Iris-TC001**	*Title*: **Delta Iris Motion** Description: This Test Case tests the delta iris motion of the camera by moving the camera with the delta command. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	GET rangeIrisLimit.0	Pass/Fail
2.	VERIFY camera supports iris limits	Pass/Fail
3.	GET systemLensFeatureControl = [systemLensFeatureControl]	Pass/Fail
4.	SET systemLensFeatureControl.0 to 0x8000	Pass/Fail
5.	SET positionIrisLens.0 to 01 <*deltaIrisMoveSpeed*> <*deltaIrisMovement*>	Pass/Fail
6.	USER VERIFY the camera lens moved towards a closed position at the movement and speed specified by the test variables *<deltaIrisMoveSpeed> and <deltaIrisMovement>*	Pass/Fail
7.	SET positionIrisLens.0 to 01 -<*deltaIrisMoveSpeed*> <*deltaIrisMovement*>	Pass/Fail
8.	USER VERIFY the camera lens moved towards an open position at the movement and speed specified by the test variables *<deltaIrisMoveSpeed> and <deltaIrisMovement>*	Pass/Fail
9.	SET systemLensFeatureControl.0 = [systemLensFeatureControl]	Pass/Fail

Test Case Results				
Tested By:		*Date Tested:*		Pass/Fail
Test Case Notes:				
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 02/06/06 Added step to turn auto iris on/off for test – RDR V1.2 02/13/06 Added test for support of iris limits – RDR V1.3 02/27/06 Implemented script and proofed – JJ			

Absolute Iris Motion

Test Case: **Iris-TC002**	*Title*:	**Absolute Iris Motion**	
	Description:	This Test Case tests the absolute iris motion of the camera by moving the camera with several different speed and direction parameters and allowing the user to verify them.	
	Variables:		
	Pass/Fail Criteria:	The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*		*Results*
1.	GET rangeIrisLimit.0		Pass/Fail
2.	VERIFY camera supports iris limits.		Pass/Fail
3.	GET systemLensFeatureControl = [systemLensFeatureControl]		Pass/Fail
4.	SET systemLensFeatureControl.0 to 0x8000		Pass/Fail
5.	SET positionIrisLens.0 to 02 *<absoluteIrisSpeed>* *<absoluteIrisPosition>*		Pass/Fail
6.	USER VERIFY the camera moved to the position defined by *<absoluteIrisPosition>*		Pass/Fail
7.	GET positionQueryIris.0		Pass/Fail
8.	VERIFY RESPONSE VALUE = *<absoluteIrisPosition>*		Pass/Fail
9.	SET positionIrisLens.0 to 02 *<absoluteIrisSpeed>* *<absoluteIrisPosition2>*		Pass/Fail
10.	USER VERIFY the camera moved to the position defined by *<absoluteIrisPosition2>*		Pass/Fail
11.	GET positionQueryIris.0		Pass/Fail
12.	VERIFY RESPONSE VALUE = *<absoluteIrisPosition2>*		Pass/Fail
13.	SET systemLensFeatureControl.0 = [systemLensFeatureControl]		Pass/Fail

Test Case Results				
Tested By:		*Date Tested:*		Pass/Fail
Test Case Notes:				
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 11/03/05 Added tests for positionQueryIris – RDR V1.2 02/06/06 Added step to turn auto iris on/off for test – RDR V1.3 02/13/06 Added test for support of iris limits – RDR V1.4 02/27/06 Implemented script and proofed – JJ			

Continuous Iris Motion with Timout

Test Case: **Iris-TC003**	*Title*: **Continuous Iris Motion with Timeout** Description: This Test Case tests the continuous iris motion of the camera by moving the camera with the continuous command using the timeout parameter to stop the camera. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	GET systemLensFeatureControl = [systemLensFeatureControl]	Pass/Fail
2.	SET systemLensFeatureControl.0 to 0x8000	Pass/Fail
3.	GET timeoutIris.0 = [timeoutIris]	Pass/Fail
4.	SET timeoutIris.0 to <*alt_contIrisTimeout*>	Pass/Fail
5.	SET positionIrisLens.0 to 03 <*conIrisSpeed*> 00 00	Pass/Fail
6.	USER VERIFY the camera lens stops moving towards a closed position after <*alt_contIrisTimeout*> milliseconds	Pass/Fail
7.	SET positionIrisLens.0 to 03 -<*conIrisSpeed*> 00 00	Pass/Fail
8.	USER VERIFY the camera lens stops moving towards an open position after <*alt_contIrisTimeout*> milliseconds	Pass/Fail
9.	SET timeoutIris.0 = [timeoutIris]	Pass/Fail
10.	SET systemLensFeatureControl.0 = [systemLensFeatureControl]	Pass/Fail
Test Case Results		
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 02/06/06 Added step to turn auto iris on/off for test – RDR V1.2 02/27/06 Implemented script and proofed – JJ	

Continuous Iris Motion with Stop

Test Case: **Iris-TC004**	*Title*: **Continuous Iris Motion with Stop** Description: This Test Case tests the continuous iris motion of the camera by moving the camera and using the stop command to stop movement. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	GET systemLensFeatureControl.0 = [systemLensFeatureControl]	Pass/Fail
2.	SET systemLensFeatureControl.0 to 0x8000	Pass/Fail
3.	GET timeoutIris.0 = [timeoutIris]	Pass/Fail
4.	SET timeoutIris.0 to 0	Pass/Fail
5.	SET positionIrisLens.0 to 03 <*conIrisSpeed*> 00 00	Pass/Fail
6.	DELAY <*alt_contIrisTimeout*> milliseconds	
7.	SET positionIrisLens.0 to 00 00 00 00	Pass/Fail
8.	USER VERIFY the camera iris stops moving to a closed position	Pass/Fail
9.	SET positionIrisLens.0 to 03 -<*conIrisSpeed*> 00 00	Pass/Fail
10.	DELAY <*alt_contIrisTimeout*> milliseconds	
11.	SET positionIrisLens.0 to 00 00 00 00	Pass/Fail
12.	USER VERIFY the camera iris stops moving to an open position	Pass/Fail
13.	SET timeoutIris.0 = [timeoutIris]	Pass/Fail
14.	SET systemLensFeatureControl.0 = [systemLensFeatureControl]	Pass/Fail
Test Case Results		
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 02/20/06 Added step to turn auto iris on/off for test – RDR V1.2 02/27/06 Implemented script and proofed – JJ	

Get and Set Label

<table>
<tr><td>Test Case:
Label-TC0001</td><td colspan="4">Title: Get and Set Label
Description: This Test Case verifies the number of labels the device can store. Test labels are stored in the device to verify storage capabilities.
Variables:
Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.</td></tr>
<tr><td>Test Step Number</td><td colspan="3">Test Procedure</td><td>Results</td></tr>
<tr><td>1.</td><td colspan="3">GET and Store labelMaximum.0 = [labelMaximum]</td><td></td></tr>
<tr><td>2.</td><td colspan="3">VERIFY RESPONSE VALUE >= <req_labelMaximum></td><td>Pass/Fail</td></tr>
<tr><td>3.</td><td colspan="3">FOR labelIndex = 1 TO [labelMaximum]</td><td></td></tr>
<tr><td>4.</td><td colspan="3">GET and Store
labelText.labelIndex = [labelText1],
labelHeight.labelIndex = [labelHeight1],
labelColor.labelIndex = [labelColor1],
labelStartRow.labelIndex = [labelStartRow1],
labelStartColumn.labelIndex = [labelStartColumn1]</td><td></td></tr>
<tr><td>5.</td><td colspan="3">SET
labelText.labelIndex = <alt_labelText>,
labelHeight.labelIndex = <alt_labelHeight>,
labelColor.labelIndex = <alt_labelColor>,
labelStartRow.labelIndex = <alt_labelStartRow>,
labelStartColumn.labelIndex = <alt_labelStartColumn></td><td>Pass/Fail</td></tr>
<tr><td>6.</td><td colspan="3">GET
labelText.labelIndex,
labelHeight.labelIndex,
labelColor.labelIndex,
labelStartRow.labelIndex,
labelStartColumn.labelIndex</td><td></td></tr>
<tr><td>7.</td><td colspan="3">VERIFY RESPONSE VALUE
labelText = <alt_labelText>,
labelHeight = <alt_labelHeight>,
labelColor = <alt_labelColor>,
labelStartRow = <alt_labelStartRow>,
labelStartColumn = <alt_labelStartColumn></td><td>Pass/Fail</td></tr>
<tr><td>8.</td><td colspan="3">SET
labelText.labelIndex = [labelText1],
labelHeight.labelIndex = [labelHeight1],
labelColor.labelIndex = [labelColor1],
labelStartRow.labelIndex = [labelStartRow1],
labelStartColumn.labelIndex = [labelStartColumn1]</td><td>Pass/Fail</td></tr>
<tr><td>9.</td><td colspan="3">NEXT labelIndex</td><td></td></tr>
<tr><td colspan="5">Test Case Results</td></tr>
<tr><td>Tested By:</td><td></td><td>Date Tested:</td><td></td><td>Pass/Fail</td></tr>
<tr><td>Test Case Notes:</td><td colspan="4"></td></tr>
<tr><td>Version History:</td><td colspan="4">V1.0 09/20/05 Initial Draft – RDR
V1.1 11/03/05 Removed deprecated labelFontType from test – RDR
V1.2 11/08/05 Removed labelStatus, Read Only object – RDR
V1.3 02/10/06 Implemented script and proofed – JJ</td></tr>
</table>

Display Camera Location

Test Case: **Label-TC0002**	*Title*: **Display Camera Location** Description: This Test Case tests the capability to display a text label on the video output. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	GET and Store labelMaximum.0 = [labelMaximum]	
2.	FOR labelIndex = 1 TO [labelMaximum]	
3.	GET and Store labelText.labelIndex = [labelText1], labelHeight.labelIndex = [labelHeight1], labelColor.labelIndex = [labelColor1], labelStartRow.labelIndex = [labelStartRow1], labelStartColumn.labelIndex = [labelStartColumn1]	
4.	SET labelText.labelIndex = "Index: labelIndex", labelHeight.labelIndex = <*alt_locLabelHeight*>, labelColor.labelIndex = <*alt_locLabelColor*>, labelStartRow.labelIndex = <*alt_locLabelStartRow*>, labelStartColumn.labelIndex = <*alt_locLabelStartColumn*>	Pass/Fail
5.	SET labelLocationLabel.0 to labelIndex	Pass/Fail
6.	SET labelEnableTextDisplay.0 to 0x80	Pass/Fail
7.	USER VERIFY that the label is displayed	Pass/Fail
8.	SET labelEnableTextDisplay.0 to 0x00	Pass/Fail
9.	USER VERIFY that the no labels are displayed	Pass/Fail
10.	SET labelEnableTextDisplay.0 to 0x80	Pass/Fail
11.	USER VERIFY that the label is displayed	Pass/Fail
12.	SET labelLocationLabel.0 to 0	Pass/Fail
13.	USER VERIFY that the label is not displayed	Pass/Fail
14.	SET labelText.labelIndex = [labelText1], labelHeight.labelIndex = [labelHeight1], labelColor.labelIndex = [labelColor1], labelStartRow.labelIndex = [labelStartRow1], labelStartColumn.labelIndex = [labelStartColumn1]	Pass/Fail
15.	NEXT labelIndex	
Test Case Results		
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 11/03/05 Removed deprecated labelFontType from test – RDR V1.2 11/08/05 Removed labelStatus, Read Only Altered labelText to change for each index – RDR V1.3 02/10/06 Implemented script and proofed – JJ	

Get Availability of Lens Equipment

Test Case: **Lens-TC001**	*Title*:	**Get Availability of Lens Equipment**		
	Description:	This Test Case identifies and verifies the availability of attached equipment to the camera.		
	Variables:			
	Pass/Fail Criteria:	The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.		
Test Step Number	*Test Procedure*			*Results*
1.	Enter the available equipment and store it to [req_systemLensEquipped]			
2.	GET systemLensEquipped.0			Pass/Fail
3.	VERIFY Response equals [req_systemLensEquipped]			Pass/Fail
Test Case Results				
Tested By:		*Date Tested:*		Pass/Fail
Test Case Notes:				
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 02/27/06 Implemented script and proofed – JJ			

Control Auto Iris

Test Case: **Lens-TC002**	*Title*:	**Control Auto Iris**		
	Description:	This Test Case enables and disables this feature while the user verifies.		
	Variables:			
	Pass/Fail Criteria:	The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.		
Test Step Number	*Test Procedure*			*Results*
1.	GET systemLensEquipped.0 = [systemLens]			Pass/Fail
2.	VERIFY that the camera supports Auto Iris			Pass/Fail
3.	GET systemLensFeatureControl.0			Pass/Fail
4.	SET systemLensFeatureControl.0 to 0x8080			Pass/Fail
5.	GET systemLensFeatureStatus.0			Pass/Fail
6.	VERIFY that bit 7 is on and Auto Iris is ON			Pass/Fail
7.	SET systemLensFeatureControl.0 to 0x8000			Pass/Fail
8.	GET systemLensFeatureStatus.0			Pass/Fail
9.	VERIFY that bit 7 is off and Auto Iris is OFF			Pass/Fail
10.	SET systemLensFeatureControl.0 = [systemLens]			Pass/Fail
Test Case Results				
Tested By:		*Date Tested:*		Pass/Fail
Test Case Notes:				
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 11/03/05 Corrected systemLensFeatureControl – RDR V1.2 02/27/06 Implemented script and proofed – JJ			

Control Auto Focus

Test Case: **Lens-TC003**	*Title*:	**Control Auto Focus**		
	Description:	This Test Case enables and disables this feature while the user verifies.		
	Variables:			
	Pass/Fail Criteria:	The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.		
Test Step Number	*Test Procedure*			*Results*
1.	GET systemLensEquipped.0 = [systemLens]			Pass/Fail
2.	VERIFY that the camera supports Auto Iris			Pass/Fail
3.	GET systemLensFeatureControl.0			Pass/Fail
4.	SET systemLensFeatureControl.0 to 0x4080			Pass/Fail
5.	GET systemLensFeatureStatus.0			Pass/Fail
6.	VERIFY that bit 7 is on and Auto Focus is ON			Pass/Fail
7.	SET systemLensFeatureControl.0 to 0x4000			Pass/Fail
8.	GET systemLensFeatureStatus.0			Pass/Fail
9.	VERIFY that bit 7 is off and Auto Focus is OFF			Pass/Fail
10.	SET systemLensFeatureControl.0 = [systemLens]			Pass/Fail
Test Case Results				
Tested By:		*Date Tested:*		Pass/Fail
Test Case Notes:				
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 11/03/05 Corrected systemLensFeatureControl – RDR V1.2 02/27/06 Implemented script and proofed – JJ			

Menu

Test Case: **Menu-TC001**	*Title*:	**Menu**
	Description:	This Test Case sends menu commands to the CCTV.
	Variables:	
	Pass/Fail Criteria:	The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.

Test Step Number	*Test Procedure*	*Results*
1.	SET menuActivate.0 to 255	Pass/Fail
2.	USER VERIFY that the menu is displayed	Pass/Fail
3.	SET menuControl.0 to 1 (pageDown)	Pass/Fail
4.	USER VERIFY that the menu moved down a page	Pass/Fail
5.	SET menuControl.0 to 2 (pageUp)	Pass/Fail
6.	USER VERIFY that the menu moved up a page	Pass/Fail
7.	SET menuControl.0 to 4 (cursorDown)	Pass/Fail
8.	USER VERIFY that the cursor moved down	Pass/Fail
9.	SET menuControl.0 to 3 (cursorUp)	Pass/Fail
10.	USER VERIFY that the cursor moved up	Pass/Fail
11.	User Input: Is the cursor in a position to move right? If Yes GOTO Step 14 If No GOTO Step 12	
12.	SET menuControl.0 to 3,4 or 9 depending on user input	Pass/Fail
13.	GOTO Step 11	
14.	SET menuControl.0 to 5 (cursorRight)	Pass/Fail
15.	USER VERIFY that the cursor moved right	Pass/Fail
16.	SET menuControl.0 to 6 (cursorLeft)	Pass/Fail
17.	USER VERIFY that the cursor moved left	Pass/Fail
18.	User Input: Is the cursor in a position to enter a value? If Yes GOTO Step 21 If No GOTO Step 19	
19.	SET menuControl.0 to 3,4,5,6 or 9, depending on user input	Pass/Fail
20.	GOTO Step 18	
21.	SET menuControl.0 to 7 (incrementValue)	Pass/Fail
22.	USER VERIFY the value has been incremented	Pass/Fail
23.	SET menuControl.0 to 8 (decrementValue)	Pass/Fail
24.	USER VERIFY the value has been decremented	Pass/Fail
25.	SET menuControl.0 to 9 (enterValue)	Pass/Fail
26.	USER VERIFY the value has been entered	Pass/Fail
27.	SET menuActivate.0 to 0	Pass/Fail
28.	USER VERIFY the menu has been deactivated	Pass/Fail
29.	SET menuActivate.0 to <*activateMenuTimeout*>	Pass/Fail
30.	USER VERIFY that the menu is displayed	Pass/Fail
31.	Delay <*activateMenuTimeout*> +1 second	
32.	USER VERIFY the menu has been deactivated	Pass/Fail

Test Case Results				
Tested By:		*Date Tested:*		Pass/Fail
Test Case Notes:				

Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 11/28/05 – Changed options in step 11 to give user more choices – JJ V1.2 02/27/06 Implemented script and proofed – JJ

Delta Pan Motion

Test Case: **Pan-TC001**	*Title*: **Delta Pan Motion** Description: This Test Case tests the delta panning motion of the camera by moving the camera with several different speed and direction parameters and allowing the user to verify them. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	GET rangePanLeftLimit.0 and rangePanRightLimit.0	Pass/Fail
2.	VERIFY camera supports panning and pan limits	Pass/Fail
3.	SET positionPan.0 to Mode: 1 (Delta), Speed: <*deltaPanMoveSpeed*>, Position: <*deltaPanMovement*>, which is hex value 01 <*deltaPanMoveSpeed*> <*deltaPanMovement*>	Pass/Fail
4.	USER VERIFY the camera moved in a clockwise direction at the movement and speed specified by <*deltaPanMovement*> and <*deltaPanMoveSpeed*>	Pass/Fail
5.	SET positionPan.0 to Mode: 1 (Delta), Speed: -<*deltaPanMoveSpeed*>, Position: <*deltaPanMovement*>, which is hex value 01 -<*deltaPanMoveSpeed*> <*deltaPanMovement*>	Pass/Fail
6.	USER VERIFY the camera moved in a counterclockwise direction at the movement and speed specified by <*deltaPanMovement*> and <*deltaPanMoveSpeed*>	Pass/Fail
Test Case Results		
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 11/11/05 Added instructions to test script to help user execute test – JJ V1.2 02/13/06 Added test for support of panning and pan limits – JJ V1.3 02/22/06 Implemented script and proofed – JJ	

Absolute Pan Motion

Test Case: **Pan-TC002**	*Title*:	**Absolute Pan Motion**
	Description:	This Test Case tests the absolute panning motion of the camera by moving the camera with several different speed and direction parameters and allowing the user to verify them.
	Variables:	
	Pass/Fail Criteria:	The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.

Test Step Number	*Test Procedure*	*Results*
1.	GET rangePanLeftLimit.0 and rangePanRightLimit.0	Pass/Fail
2.	VERIFY camera supports panning and pan limits	Pass/Fail
3.	GET rangePanHomePosition.0 = [rangePanHomePostion]	Pass/Fail
4.	SET positionPan.0 to Mode:2 (Absolute), Speed: <*absolutePanSpeed*>, Position: [rangePanHomePostion], which is hex value 02 <*absolutePanSpeed*> [rangePanHomePostion]	Pass/Fail
5.	Note that current position as the Home position	
6.	SET positionPan.0 to Mode: 2(Absolute), Speed: <*absolutePanSpeed*>, Position: <*absolutePanPosition*>, which is hex value 02 <*absolutePanSpeed*> <*absolutePanPosition*>*.*	Pass/Fail
7.	USER VERIFY the camera moved to the position <*absolutePanPosition*>	Pass/Fail
8.	GET positionQueryPan.0	Pass/Fail
9.	VERIFY RESPONSE VALUE = <*absolutePanPosition*>	Pass/Fail
10.	SET positionPan.0 to Mode: 2(Absolute), Speed: <*absolutePanSpeed*>, Position: [rangePanHomePostion], which is hex value 02 <*absolutePanSpeed*> [rangePanHomePostion]	Pass/Fail
11.	USER VERIFY the camera moved back to Home Position	Pass/Fail
12.	SET positionPan.0 to Mode: 2(Absolute), Speed: <*absolutePanSpeed*>, Position: <*absolutePanPosition2*>, which is hex value 02 <*absolutePanSpeed*> <*absolutePanPosition2*>*.*	Pass/Fail
13.	USER VERIFY the camera moved to the position <*absolutePanPosition2*>	Pass/Fail
14.	GET positionQueryPan.0	Pass/Fail
15.	VERIFY RESPONSE VALUE = <*absolutePanPosition2*>	Pass/Fail

Test Case Results

Tested By:		*Date Tested:*		Pass/Fail
Test Case Notes:				
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 11/04/05 Added test for positionQueryPan – JJ V1.2 11/11/05 Added steps to set position to Home position to test absolute movements – JJ V1.3 01/30/06 Added step to move back to Home after first absolute test – JJ V1.4 02/13/06 Added test for support of panning and pan limits – JJ V1.5 02/22/06 Implemented script and proofed – JJ			

Continuous Pan Motion with Timeout

Test Case: **Pan-TC003**	*Title*: **Continuous Pan Motion with Timeout** Description: This Test Case tests the continuous panning motion of the camera by moving the camera with the continuous command using the timeout parameter to stop the camera. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	GET rangePanLeftLimit.0 and rangePanRightLimit.0	Pass/Fail
2.	VERIFY camera supports panning and pan limits	Pass/Fail
3.	GET timeoutPan.0 = [timeoutPan]	Pass/Fail
4.	SET timeoutPan.0 to <*alt_contPanTimeout*>	Pass/Fail
5.	SET positionPan.0 to Mode: 3(Continuous), Speed: <*conPanSpeed*>, Position: 0, which is hex value 03 <*conPanSpeed*> 00 00	Pass/Fail
6.	USER VERIFY the camera stops moving in a clockwise direction after <*alt_contPanTimeout*> milliseconds	Pass/Fail
7.	SET positionPan.0 to Mode: 3(Continuous), Speed: -<*conPanSpeed*>, Position: 0, which is hex value 03 -<*conPanSpeed*> 00 00	Pass/Fail
8.	USER VERIFY the camera stops moving in a counterclockwise direction after <*alt_contPanTimeout*> milliseconds	Pass/Fail
9.	SET timeoutPan.= [timeoutPan]	Pass/Fail
Test Case Results		
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 11/11/05 Added instructions to test script to help user execute test – JJ V1.2 02/13/06 Added test for support of panning and pan limits – JJ V1.3 02/22/06 Implemented script and proofed – JJ	

Continuous Pan Motion with Stop

Test Case: **Pan-TC004**	*Title*: **Continuous Pan Motion with Stop** Description: This Test Case tests the continuous panning motion of the camera by moving the camera and using the stop command to stop movement. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	GET rangePanLeftLimit.0 and rangePanRightLimit.0	Pass/Fail
2.	VERIFY camera supports panning and pan limits	Pass/Fail
3.	GET timeoutPan.0 = [timeoutPan]	Pass/Fail
4.	SET timeoutPan.0 to 0	Pass/Fail
5.	SET positionPan.0 to Mode: 3(Continuous), Speed: <*conPanSpeed*>, Position: 0, which is hex value 03 <*conPanSpeed*> 00 00	Pass/Fail
6.	DELAY <*alt_contPanTimeout*> milliseconds	
7.	SET positionPan.0 to 00 00 00 00	Pass/Fail
8.	USER VERIFY the camera stops moving	Pass/Fail
9.	SET positionPan.0 to Mode: 3(Continuous), Speed: -<*conPanSpeed*>, Position: 0, which is hex value 03 -<*conPanSpeed*> 00 00	Pass/Fail
10.	DELAY <*alt_contPanTimeout*> milliseconds	
11.	SET positionPan.0 to 00 00 00 00	Pass/Fail
12.	USER VERIFY the camera stops moving	Pass/Fail
13.	SET timeoutPan.0 = [timeoutPan]	Pass/Fail
Test Case Results		
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 02/13/06 Added test for support of panning and pan limits – JJ V1.2 02/22/06 Implemented script and proofed – JJ	

Change Administrator Community Name

Test Case: **Security-TC0001**	*Title*: **Change Administrator Community Name** Description: This Test Case verifies that the administrator can change the administrator community name stored in the DUT and that the change properly affects the operation of the device. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	As a part of the project-specific Test Plan, define a <*temp_communityName*> to use for testing the administrator community name. This must be a different name than the current communityNameAdmin and communityNameUser(s) values, which will be referred to as the original community name. *Note:* Valid administrator community names are between 8 and 16 characters long, inclusive and may contain any ASCII character, including non-printable characters.	
2.	CONFIGURE COMMUNITY NAME OUT = <current_communityName>	
3.	GET communityNameAdmin.0 = [communityNameAdmin]	Pass/Fail
4.	VERIFY RESPONSE VALUE= COMMUNITY NAME OUT	Pass/Fail
5.	SET communityNameAdmin.0 = <*temp_communityName*>	Pass/Fail
6.	GET communityNameAdmin.0 and VERIFY that either: 1. No response is received or 2. The RESPONSE ERROR is 2 (noSuchName) and the RESPONSE INDEX is 1	Pass/Fail
7.	CONFIGURE COMMUNITY NAME OUT = <*temp_communityName*>	
8.	GET communityNameAdmin.0	Pass/Fail
9.	VERIFY RESPONSE VALUE = COMMUNITY NAME OUT	Pass/Fail
10.	SET communityNameAdmin.0 = [communityNameAdmin]	Pass/Fail
11.	GET communityNameAdmin.0 and VERIFY that either: 1. No response is received or 2. The RESPONSE ERROR is 2 (noSuchName) and the RESPONSE INDEX is 1	Pass/Fail
12.	CONFIGURE COMMUNITY NAME OUT = [communityNameAdmin]	
13.	GET communityNameAdmin.0	Pass/Fail
14.	VERIFY RESPONSE VALUE = COMMUNITY NAME OUT	Pass/Fail
Test Case Results		
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 02/13/06 Original as defined in NTCIP 8007 v01.20 - B.3.4 Change Administrator Community Name – RDR V2.0 02/15/06 Changed “Change” to CONFIGURE Removed “Temporary community name” from Variables field and adopted convention of using [name of variable] for the text description of local variables and using <name> for externally defined constants – RDR V2.1 02/22/06 Implemented script and proofed – JJ	

Change User Community Name

Test Case: **Security-TC002**	*Title*: **Change User Community Name** Description: This Test Case verifies that the administrator can change the user community names and their masks stored in the DUT and that the changes properly affect the operation of the device. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	PRE-CONDITION: Ensure that the DUT is configured such that the user community names are something other than “NTCIP USER #”, where # is the row number in the community name table. When set to 0, the communityNameAccessMask effectively changes the any object with read-write access to ready only. The object to test is <maskReadWriteOID>. The default value of the OID in the set up file is sysName.0. However, one may replace these with any other object OID supported by the DUT that that has similar characteristics and is read-write.	
2.	GET globalMaxModules.0 *Note:* This object is referenced throughout this procedure. A Test Plan may replace all references to this object in this Test Case with a reference to any other object supported by the DUT that will have a constant value for the duration of the Test Case. The globalMaxModules is used in this procedure because most every device supports it.	Pass/Fail
3.	RECORD the RESPONSE VALUE = [globalMaxModules]	
4.	GET communityNamesMax.0	Pass/Fail
5.	RECORD the RESPONSE VALUE = [communityNamesMax]	
6.	VERIFY RESPONSE VALUE >= <req_communityNamesMax>	Pass/Fail
7.	FOR N = 1 TO [communityNamesMax] *Note:* This loop tests the original community names, to make sure that they work and then SETs them to new values and ensures that the new values work.	
8.	GET the following objects: communityNameUser.N, communityNameAccessMask.N	Pass/Fail
9.	RECORD the RESPONSE VALUES = [communityNameUser(N)] and [communityNameAccessMask(N)] for each object indicating which instance of N is being used	
10.	CONFIGURE COMMUNITY NAME OUT = [communityNameUser(N)]	
11.	GET globalMaxModules.0	Pass/Fail
12.	VERIFY RESPONSE VALUE = [globalMaxModules]	Pass/Fail
13.	GET <maskReadWriteOID>	Pass/Fail
14.	RECORD the RESPONSE VALUE = [temp_maskReadWriteOIDValue]	
15.	SET <maskReadWriteOID> =	Pass/Fail

	[temp_maskReadWriteOIDValue] *Note:* This ensures that communityNameAccessMask.N is not blocking the normal read-write access of <maskReadWriteOID>.	
16.	CONFIGURE COMMUNITY NAME OUT = administrator community name	
17.	SET the following objects to the values as shown: communityNameUser.N to the string "NTCIP USER #" where # is the row number N, communityNameAccessMask.N to the value of zero (0)	Pass/Fail
18.	GET the following objects: communityNameUser.N, communityNameAccessMask.N	Pass/Fail
19.	VERIFY RESPONSE VALUE communityNameUser.N = "NTCIP USER #"	Pass/Fail
20.	VERIFY that the RESPONSE VALUE communityNameAccessMask.N = 0	Pass/Fail
21.	Change the COMMUNITY NAME OUT to "NTCIP USER #" where # is the row number N	
22.	GET globalMaxModules.0	Pass/Fail
23.	VERIFY RESPONSE VALUE = [globalMaxModules]	Pass/Fail
24.	SET <maskReadWriteOID> = <maskReadWriteOIDValue> and VERIFY RESPONSE ERROR is 2 (noSuchName) and the RESPONSE INDEX is 1 *Note:* This ensures that changing the access mask to 0 SETs the access of <maskReadWriteOID> = readOnly.	Pass/Fail
25.	CONFIGURE the COMMUNITY NAME OUT to the administrator community name	
26.	NEXT N	
27.	FOR N = 1 TO [communityNamesMax] *Note:* This loop tests the original community names, to make sure that they no longer work.	
28.	CONFIGURE COMMUNITY NAME OUT = [communityNameUser(N)] previously recorded	
29.	GET globalMaxModules.0 and VERIFY that either: 1. No response is received or 2. The RESPONSE ERROR is 2 (noSuchName) and the RESPONSE INDEX is 1	Pass/Fail
30.	NEXT N	
31.	FOR N = 1 TO [communityNamesMax] *Note:* This loop tests the original community names once restored to the DUT to make sure that they work.	
32.	CONFIGURE COMMUNITY NAME OUT to the <original administrator community name	
33.	SET the following objects to the values as shown: communityNameUser.N = [communityNameUser(N)], communityNameAccessMask.N = [communityNameAccessMask(N)]	Pass/Fail
34.	GET the following objects: communityNameUser.N, communityNameAccessMask.N	Pass/Fail

35.	VERIFY RESPONSE VALUE communityNameUser.N = [communityNameUser(N)]	Pass/Fail
36.	VERIFY RESPONSE VALUE communityNameAccessMask.N = [communityNameAccessMask(N)]	Pass/Fail
37.	Change the COMMUNITY NAME OUT to the [communityNameUser(N)]	
38.	GET globalMaxModules.0	Pass/Fail
39.	VERIFY RESPONSE VALUE = [globalMaxModules]	Pass/Fail
40.	NEXT N	
41.	FOR N = 1 TO [communityNamesMax] *Note:* This loop tests the temporary community names created during this Test Case to make sure that they no longer work.	
42.	CONFIGURE COMMUNITY NAME OUT = "NTCIP USER #", where # is the row number N	
43.	GET globalMaxModules.0 and VERIFY that either: 1. No response is received or 2. The RESPONSE ERROR is 2 (noSuchName) and the RESPONSE INDEX is 1	Pass/Fail
44.	NEXT N	
45.	CONFIGURE COMMUNITY NAME OUT = administrator community name	
46.	PRE-CONDITION: Ensure that the DUT is configured such that the user community names are something other than “NTCIP USER #”, where # is the row number in the community name table. When set to 0, the communityNameAccessMask effectively changes the any object with read-write access to ready only. The object to test is <maskReadWriteOID>. The default value of the OID in the setup file is sysName.0. However, one may replace these with any other object OID supported by the DUT that has similar characteristics and is read-write.	

Test Case Results				
Tested By:		*Date Tested:*		Pass/Fail
Test Case Notes:	If the Test Case fails, one or more of the user community names in the DUT may be “NTCIP USER#”, where # is the row number in the user community name table.			
Version History:	V1.0 02/13/06 Original as defined in NTCIP 8007 v01.20 - B.3.6 Change User Community Name V2.0 02/15/06 Reformatted to use FOR and NEXT and simplified a number of steps. High-lighted potential invalid response to GET of globalMaxModules.0 Added last step to restore COMMUNITY NAME OUT Add steps to check that setting of communityNameAccessMasks = 0 changes access to read-only. Changed “Change” to CONFIGURE Removed “Temporary community name” from Variables field and adopted convention of using [name of variable] for the text description of local variables and using <name> for externally defined constants. – RDR V2.1 02/22/06 Implemented script and proofed – JJ			

Delta Tilt Motion

Test Case: **Tilt-TC001**	*Title*: **Delta Tilt Motion** Description: This Test Case tests the delta tilt motion of the camera by moving the camera with several different speed and direction parameters and allowing the user to verify them. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.			
Test Step Number	*Test Procedure*			*Results*
1.	GET rangeTiltDownLimit.0 and rangeTiltUpLimit.0			Pass/Fail
2.	VERIFY camera supports tilt limits			Pass/Fail
3.	SET positionTilt.0 to Mode: 1 (Delta), Speed: <*deltaTitlMoveSpeed*>, Position: <*deltaTiltMovement*>, which is hex value 01 <*deltaTitlMoveSpeed*> <*deltaTiltMovement*>			Pass/Fail
4.	USER VERIFY the camera moved in a up direction at the movement and speed specified by <*deltaTiltMoveSpeed*> *and* <*deltaTiltMovement*>			Pass/Fail
5.	SET positionTilt.0 to Mode: 1 (Delta), Speed: -<*deltaTiltMoveSpeed*>, Position: <*deltaTiltMovement*>, which is hex value 01 -<*deltaTiltMoveSpeed*> <*deltaTiltMovement*>			Pass/Fail
6.	USER VERIFY the camera moved in a down direction at the movement and speed specified by <*deltaTiltMoveSpeed*> *and* <*deltaTiltMovement*>			Pass/Fail
Test Case Results				
Tested By:		*Date Tested:*		Pass/Fail
Test Case Notes:				
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 02/13/06 Added test for support of tilt limits – JJ V1.2 02/22/06 Implemented script and proofed – JJ			

Absolute Tilt Motion

Test Case: **Tilt-TC002**	*Title*: **Absolute Tilt Motion** Description: This Test Case tests the absolute tilt motion of the camera by moving the camera with several different speed and direction parameters and allowing the user to verify them. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	GET rangeTiltDownLimit.0 and rangeTiltUpLimit.0	Pass/Fail
2.	VERIFY camera supports tilt limits	Pass/Fail
3.	SET positionTilt.0 to Mode: 2 (Absolute), Speed: 0, Position: 0, which is hex value 02 00 00 00	Pass/Fail
4.	SET positionTilt.0 to Mode: 2 (Absolute), Speed: <*absoluteTiltSpeed*>, Position: <*absoluteTiltPosition*>, which is hex value 02 <*absoluteTiltSpeed*> <*absoluteTiltPosition*>	Pass/Fail
5.	USER VERIFY the camera moved to the position defined by <*absoluteTiltPosition*>	Pass/Fail
6.	GET positionQueryTilt.0	Pass/Fail
7.	VERIFY RESPONSE VALUE = <*absoluteTiltPosition*>	Pass/Fail
8.	SET positionTilt.0 to Mode: 2 (Absolute), Speed: 0, Position: 0, which is hex value 02 00 00 00	Pass/Fail
9.	SET positionTilt.0 to Mode: 2 (Absolute), Speed: <*absoluteTiltSpeed*>, Position: <*absoluteTiltPosition2*>, which is hex value 02 <*absoluteTiltSpeed*> <*absoluteTiltPosition2*>	Pass/Fail
10.	USER VERIFY the camera moved to the position defined by <*absoluteTiltPosition2*>	Pass/Fail
11.	GET positionQueryTilt.0	Pass/Fail
12.	VERIFY RESPONSE VALUE = <*absoluteTiltPosition2*>	Pass/Fail
13.	SET positionTilt.0 to Mode: 2 (Absolute), Speed: 0, Position: 0, which is hex value 02 00 00 00	Pass/Fail
Test Case Results		
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 11/03/05 Added test for positionQueryTilt – JJ V1.2 02/03/06– Added step to set position to Home (zero) before testing absolute movement – JJ V1.3 02/13/06 Added test for support of tilt limits – JJ V1.4 02/22/06 Implemented script and proofed – JJ	

Continuous Tilt Motion with Timeout

Test Case: **Tilt-TC003**	*Title*: **Continuous Tilt Motion with Timeout** Description: This Test Case tests the continuous tilt motion of the camera by moving the camera with the continuous command using the timeout parameter to stop the camera. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	GET timeoutTilt.0 = [timeoutTilt]	Pass/Fail
2.	SET timeoutTilt.0 to <*alt_contTiltTimeout*>	Pass/Fail
3.	SET positionTilt.0 to Mode: 3 (Continuous), Speed: <*conTiltSpeed*>, Position: 0, which is hex value 03 <*conTiltSpeed*> 00 00	Pass/Fail
4.	USER VERIFY the camera moves up and stops after <*alt_contTiltTimeout*> milliseconds	Pass/Fail
5.	SET positionTilt.0 to Mode: 3 (Continuous), Speed: -<*conTiltSpeed*>, Position: 0, which is hex value 03 -<*conTiltSpeed*> 00 00	Pass/Fail
6.	USER VERIFY the camera moves down and stops after <*alt_contTiltTimeout*> milliseconds	Pass/Fail
7.	SET timeoutTilt.0 = [timeoutTilt]	Pass/Fail

Test Case Results				
Tested By:		*Date Tested:*		Pass/Fail
Test Case Notes:				
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 02/22/06 Implemented script and proofed – JJ			

Continuous Tilt Motion with Stop

Test Case: **Tilt-TC0004**	*Title*: **Continuous Tilt Motion with Stop** Description: This Test Case tests the continuous tilting motion of the camera by moving the camera and using the stop command to stop movement. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	GET timeoutTilt.0 = [timeoutTilt]	Pass/Fail
2.	SET timeoutTilt.0 to 0	Pass/Fail
3.	SET positionTilt.0 to Mode: 3 (Continuous), Speed: <*conTiltSpeed*>, Position: 0, which is hex value 03 <*conTiltSpeed*> 00 00	Pass/Fail
4.	DELAY <*alt_contTiltTimeout*> milliseconds	
5.	SET positionTilt.0 to Mode: 0, Speed: 0, Position: 0, which is hex value 00 00 00 00	Pass/Fail
6.	USER VERIFY the camera stops moving	Pass/Fail
7.	SET positionTilt.0 to Mode: 3 (Continuous), Speed: -<*conTiltSpeed*>, Position: 0, which is hex value 03 -<*conTiltSpeed*> 00 00	Pass/Fail
8.	DELAY <*alt_contTiltTimeout*> milliseconds	
9.	SET positionTilt.0 to Mode: 0, Speed: 0, Position: 0, which is hex value 00 00 00 00	Pass/Fail
10.	USER VERIFY the camera stops moving	Pass/Fail
11.	SET timeoutTilt.0 = [timeoutTilt]	Pass/Fail
Test Case Results		
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 02/22/06 Implemented script and proofed – JJ	

Preset Position

Test Case: **Zone-TC001**	*Title*:	**Preset Position**		
	Description:	This Test Case stores and moves the camera to preset camera positions.		
	Variables:			
	Pass/Fail Criteria:	The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.		
Test Step Number	*Test Procedure*			*Results*
1.	SET positionPan.0 to 02 <*presetMovementSpeed*> <*presetPanPosition1*> and positionTilt.0 to 02 <*presetMovementSpeed*> <*presetTiltPosition1*>			Pass/Fail
2.	Note current camera position as position 1.			
3.	SET presetStorePosition.0 to <*presetStore1*>			Pass/Fail
4.	SET positionPan.0 to 02 <*presetMovementSpeed*> <*presetPanPosition2*> and positionTilt.0 to 02 <*presetMovementSpeed*> <*presetTiltPosition2*>			Pass/Fail
5.	Note current camera position as position 2.			
6.	SET presetStorePosition.0 to <*presetStore2*>			Pass/Fail
7.	SET presetGotoPosition.0 to <*presetStore1*>			Pass/Fail
8.	USER VERIFY the camera moved to position 1			Pass/Fail
9.	GET presetPositionQuery.0			Pass/Fail
10.	VERIFY RESPONSE VALUE = 1			Pass/Fail
11.	SET presetGotoPosition.0 to <*presetStore2*>			Pass/Fail
12.	USER VERIFY the camera moved to position 2			Pass/Fail
13.	GET presetPositionQuery.0			Pass/Fail
14.	VERIFY RESPONSE VALUE = 2			Pass/Fail
Test Case Results				
Tested By:		*Date Tested:*		Pass/Fail
Test Case Notes:				
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 11/03/05 Added test for presetPositionQuery.0 – JJ V1.2 02/27/06 Implemented script and proofed – JJ			

Get-Set Zone

Test Case: **Zone-TC002**	*Title*: **Get-Set Zone** Description: This Test Case tests the storage of camera zones. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	GET zoneMaximum.0 = [zoneMaximum]	Pass/Fail
2.	VERIFY zoneMaximum.0 >= <*req_ZoneMaximum*>	Pass/Fail
3.	FOR zoneIndex = 1 to [zoneMaximum]	
4.	GET zoneLabel.zoneIndex = [zonelabel], zonePanRightLimit.zoneIndex = [zonePanRightLimit], zoneTiltUpLimit.zoneIndex = [zoneTiltUpLimit], zoneTiltDownLimit.zoneIndex = [zoneTiltDownLimit], zonePanLeftLimit.zoneIndex = [zonePanLeftLimit]	Pass/Fail
5.	SET zoneLabel.zoneIndex = <*alt_zoneLabel*>, zonePanRightLimit.zoneIndex = <*alt_zonePanRightLimit*>, zoneTiltUpLimit.zoneIndex = <*alt_zoneTiltUpLimit*>, zoneTiltDownLimit.zoneIndex = <*alt_zoneTiltDownLimit*>, zonePanLeftLimit.zoneIndex = <*alt_zonePanLeftLimit*>	Pass/Fail
6.	SET zoneLabel.zoneIndex = [zonelabel], zonePanRightLimit.zoneIndex = [zonePanRightLimit], zoneTiltUpLimit.zoneIndex = [zoneTiltUpLimit], zoneTiltDownLimit.zoneIndex = [zoneTiltDownLimit], zonePanLeftLimit.zoneIndex = [zonePanLeftLimit]	Pass/Fail
7.	NEXT zoneIndex	
	Test Case Results	
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 02/27/06 Implemented script and proofed – JJ	

Move In and Out of Zone

Test Case: **Zone-TC003**	*Title*: **Move In and Out of Zone** Description: This Test Case tests the labeling capability of zones by moving to areas within zones. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	GET zoneLabel.<*indexZone*> = [zoneLabel] zonePanRightLimit.<*indexZone*> = [zonePanRightLimit], zoneTiltUpLimit.<*indexZone*> = [zoneTiltUpLimit], zoneTiltDownLimit.<*indexZone*> = [zoneTiltDownLimit],	Pass/Fail

	zonePanLeftLimit. <*indexZone*> = [zonePanLeftLimit]	
2.	SET zoneLabel. <*indexZone*>= <*alt_zoneLabel*>, zonePanRightLimit.<*indexZone*>= <*alt_zonePanRightLimit*>, zoneTiltUpLimit.<*indexZone*>= <*alt_zoneTiltUpLimit*>, zoneTiltDownLimit.<*indexZone*>= <*alt_zoneTiltDownLimit*>, zonePanLeftLimit.<*indexZone*>= <*alt_zonePanLeftLimit*>	Pass/Fail
3.	GET labelText.< *indexZone* > = [labelText], labelHeight.< *indexZone* > = [labelHeight], labelColor.< *indexZone* > = [labelColor], labelStartRow.< *indexZone* > = [labelStartRow], labelStartColumn.< *indexZone* > = [labelStartColumn], labelStatus.< *indexZone* > = [labelStatus]	Pass/Fail
4.	SET labelText.< *indexZone* > = <*alt_ZlabelText*>, labelHeight.< *indexZone* > = <*alt_ZlabelHeight*>, labelColor.< *indexZone* > = <*alt_ZlabelColor*>, labelStartRow.< *indexZone* > = <*alt_ZlabelStartRow*>, labelStartColumn.< *indexZone* > = <*alt_ZlabelStartColumn*>, labelStatus.< *indexZone* > = <*alt_ZlabelStatus*>	Pass/Fail
5.	SET positionPan.0 to 02 <*zoneMoveSpeed*> (<*alt_zoneRightLimit*>-1) and positionTilt.0 to 02 <*zoneMoveSpeed*> (<*alt_zoneUpLimit*>-1)	Pass/Fail
6.	USER VERIFY that the label for the current zone is correctly displayed, as set in step 2	Pass/Fail
7.	SET positionPan.0 to 02 <*zoneMoveSpeed*> (<*alt_zoneRightLimit*>+1) and positionTilt.0 to 02 <*zoneMoveSpeed*> (<*alt_zoneUpLimit*>+1)	Pass/Fail
8.	USER VERIFY that the label for the current zone is not displayed	Pass/Fail
9.	SET zoneLabel.<*indexZone*> = [zoneLabel], zonePanRightLimit.<*indexZone*> = [zonePanRightLimit], zoneTiltUpLimit.<*indexZone*> = [zoneTiltUpLimit], zoneTiltDownLimit.<*indexZone*> = [zoneTiltDownLimit], zonePanLeftLimit.<*indexZone*> = [zonePanLeftLimit]	Pass/Fail
10.	SET labelText.< *indexZone* > = [labelText], labelHeight.< *indexZone* > = [labelHeight], labelColor.< *indexZone* > = [labelColor], labelStartRow.< *indexZone* > = [labelStartRow], labelStartColumn.< *indexZone* > = [labelStartColumn], labelStatus.< *indexZone* > = [labelStatus]	Pass/Fail

Test Case Results				
Tested By:		*Date Tested:*		Pass/Fail
Test Case Notes:				
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 11/03/05 Removed labelFontType, deprecated – JJ V1.2 02/27/06 Implemented script and proofed – JJ			

Delta Zoom Motion

Test Case: **Zoom-TC001**	*Title*: **Delta Zoom Motion** Description: This Test Case tests the delta zoom motion of the camera by moving the camera with several different speed and direction parameters and allowing the user to verify them. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	GET rangeZoom.0	Pass/Fail
2.	VERIFY camera supports zoom limits	Pass/Fail
3.	SET positionZoomLens.0 to Mode: 1(Delta), Speed: <*deltaZoomMoveSpeed*>, Position: <*deltaZoomMovement*>, which is hex value 01 <*deltaZoomMoveSpeed*> <*deltaZoomMovement*>	Pass/Fail
4.	USER VERIFY the camera lens moved towards a telephoto position at the movement and speed specified by the test variables <*deltaZoomMoveSpeed*> and <*deltaZoomMovement*>	Pass/Fail
5.	SET positionZoomLens.0 to Mode: 1(Delta), Speed: -<*deltaZoomMoveSpeed*>, Position: <*deltaZoomMovement*>, which is hex value 01 -<*deltaZoomMoveSpeed*> <*deltaZoomMovement*>	Pass/Fail
6.	USER VERIFY the camera lens moved towards a wide angle position at the movement and speed specified by the test variables <*deltaZoomMoveSpeed*> and <*deltaZoomMovement*>	Pass/Fail
	Test Case Results	
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 02/13/06 Added test for support of zoom limits – JJ V1.2 02/27/06 Implemented script and proofed – JJ	

Absolute Zoom Motion

Test Case: **Zoom-TC002**	*Title*: **Absolute Zoom Motion** Description: This Test Case tests the absolute zoom motion of the camera by moving the camera with several different speed and direction parameters and allowing the user to verify them. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	GET rangeZoomt.0	Pass/Fail
2.	VERIFY camera supports zoom limits	Pass/Fail
3.	SET positionZoomLens.0 to 02 00 00 00	Pass/Fail
4.	SET positionZoomLens.0 to 02 <*absoluteZoomSpeed*> <*absoluteZoomPosition*>	Pass/Fail
5.	VERIFY the camera moved to the position defined by <*absoluteZoomPosition*>	Pass/Fail

6.	GET positionQueryZoom.0	Pass/Fail
7.	VERIFY RESPONSE VALUE = <*absoluteZoomPosition*>	Pass/Fail
8.	SET positionZoomLens.0 to 02 00 00 00	Pass/Fail
9.	SET positionZoomLens.0 to 02 <*absoluteZoomSpeed*> <*absoluteZoomPosition2*>	Pass/Fail
10.	VERIFY the camera moved to the position defined by <*absoluteZoomPosition2*>	Pass/Fail
11.	GET positionQueryZoom.0	Pass/Fail
12.	VERIFY RESPONSE VALUE = <*absoluteZoomPosition2*>	Pass/Fail
Test Case Results		
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 11/03/05 Added tests for positionQueryZoom – JJ V1.2 02/13/06 Added steps to set position to Home position to test absolute movements. Added test for support of zoom limits – JJ V1.3 02/27/06 Implemented script and proofed – JJ	

Continuous Zoom Motion with Timeout

Test Case: **Zoom-TC003**	*Title*: **Continuous Zoom Motion with Timeout** Description: This Test Case tests the continuous zoom motion of the camera by moving the camera with the continuous command using the timeout parameter to stop the camera. Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1.	GET timeoutZoom.0 = [timeoutZoom]	Pass/Fail
2.	SET timeoutZoom.0 to <*alt_contZoomTimeout*>	Pass/Fail
3.	SET positionZoomLens.0 to 03 <*conZoomSpeed*> 00 00	Pass/Fail
4.	USER VERIFY the camera lens stops moving towards a telephoto position after <*alt_contZoomTimeout*> milliseconds	Pass/Fail
5.	SET positionZoomLens.0 to 03 -<*conZoomSpeed*> 00 00	Pass/Fail
6.	USER VERIFY the camera lens stops moving towards a wide angle position after <*alt_contZoomTimeout*> milliseconds	Pass/Fail
7.	SET timeoutZoom.0 = [timeoutZoom]	Pass/Fail
Test Case Results		
Tested By:	*Date Tested:*	Pass/Fail
Test Case Notes:		
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 02/27/06 Implemented script and proofed – JJ	

Continuous Zoom Motion with Stop

Test Case: **Zoom-TC004**	*Title*:	**Continuous Zoom Motion with Stop**	
	Description:	This Test Case tests the continuous zooming motion of the camera by moving the camera and using the stop command to stop movement.	
	Variables:		
	Pass/Fail Criteria:	The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*		*Results*
1.	GET timeoutZoom.0 = [timeoutZoom]		Pass/Fail
2.	SET timeoutZoom.0 to 0		Pass/Fail
3.	SET positionZoomLens.0 to 03 <*conZoomSpeed*> 00 00		Pass/Fail
4.	DELAY <*alt_contZoomTimeout*> milliseconds		
5.	SET positionZoomLens.0 to 00 00 00 00		Pass/Fail
6.	USER VERIFY the camera stops moving		Pass/Fail
7.	SET positionZoomLens.0 to 03 -<*conZoomSpeed*> 00 00		Pass/Fail
8.	DELAY <*alt_contZoomTimeout*> milliseconds		
9.	SET positionZoomLens.0 to 00 00 00 00		Pass/Fail
10.	USER VERIFY the camera stops moving		Pass/Fail
11.	SET timeoutZoom.0= [timeoutZoom]		Pass/Fail
Test Case Results			
Tested By:		*Date Tested:*	Pass/Fail
Test Case Notes:			
Version History:	V1.0 09/20/05 Initial Draft – RDR V1.1 02/27/06 Implemented script and proofed – JJ		

REFERENCES FOR APPENDIX C

1. NTCIP 1205 – Object Definitions for Closed Circuit Television (CCTV) Camera Control, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=1205. Accessed August 17, 2006.

2. NTCIP 1201 – Global Object Definitions, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=1201. Accessed August 17, 2006.

3. Test Procedures for NTCIP-conformant Closed Circuit Television (CCTV) Camera Controllers, Enterprise Consortium. http://enterprise.prog.org/ (document no longer available) Accessed December 2002.

4. NTCIP 8007 – Testing and Conformity Assessment Documentation within NTCIP Standards Publications, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=8007. Accessed August 17, 2006.

APPENDIX D:
CCTV
PROTOCOL IMPLEMENTATION CONFORMANCE SPECIFICATION

INTRODUCTION

This appendix contains a Protocol Implementation Conformance Specification (PICS) for a specific camera and controller that was tested using the Prequalification Test Case - TC001 appearing in Appendix C. By adding indications that an implementation has support for a feature, an NTCIP Profile Requirements List (PRL) becomes a PICS. In the context of NTCIP testing, the PRL provides a standardized format to record test results that check for object support, maximum or limit values, index values, and the range of supported values.

NTCIP 1205 - Object Definitions for Closed Circuit Television (CCTV) Camera Control (NTCIP 1205-CCTV) dictates the format of the PICS (*1*). NTCIP 1205-CCTV grants specific copyright permission to use Annex B of that standard for creating a PICS (*1*). Except for the object support and supported values column entries, the format comes from the original PRL as it appears in the standard. The researcher indicate changes made in Amendment 1 v10 versus NTCIP 1205 v01.08 by highlighting them in bold-italics (*1*).

INTERPRETING RESULTS

For use in reporting test results, the words in all uppercase letters in the object support column indicate the results of specific object tests. A “YES” in the support column indicates that a retrieval operation did not return an error and, therefore, has support. Words appearing in mixed case in the object support column (for example, “Yes” or “Yes / No”) indicate the absence of any specific test and appear as in the original. A single value in the supported values column (for example, “64”) indicates a constant value supplied by the implementation. A set of passed or failed values in the supported values column (for example, “PASSED: 0, 1, FAILED: -1,” indicates the implementation’s responses to test values that were sent to the object. The test script that implements the Prequalification Test Case records the results directly onto the form.

Annex B
INFORMATION PROFILE

(Informative)[1]

Notice – PRL excerpted from a draft document containing preliminary information that is subject to change. Object names in bold-italics have additional information indicating changes between Version 01.08 and Amendment 1.

A Conformance Group is a basic unit of conformance and is used to specify a collection of related managed objects. The Conformance Group designation applied to a set of objects provides a systematic means for determining which objects are required to support a function. If a device has multiple functions, a Conformance Group will be defined for each function. Conformance Group definitions will be found in the NTCIP Object Definition Standard documents. The Object Definition Standard may define a Conformance Group with objects that are not in lexicographic order and only apply to devices of that type.

The related managed objects of a Conformance Group may include mandatory and/or optional objects. Mandatory objects within a Conformance Group shall be implemented. Optional objects shall be implemented only if a defined function of the device requires that particular object.

For example, assume a device implements an asynchronous RS-232 interface. It must implement all the mandatory objects in the Asynchronous Conformance Group of the RS-232 MIB. It would not have to implement the Synchronous Conformance Group of objects unless it also provided a synchronous interface.

Assume also that the Asynchronous Conformance Group has a *CRC error counter* object that is optional. The *CRC error counter* object would not have to be implemented unless the device used CRC checking on the asynchronous interface.

Conformance Groups are defined as either mandatory or optional. If a Conformance Group is mandatory, all of the objects with STATUS "mandatory" that are part of the Conformance Group shall be present for a device to claim conformance to the Conformance Group. If a Conformance Group is optional, all of the objects that are part of the Conformance Group with the STATUS "mandatory" shall be present if the device supports the Conformance Group. Objects with the STATUS "optional" may be supported.

When a table is included in a Conformance Group, all objects contained in the table are included by reference. This is because a table is defined as a SEQUENCE OF {SEQUENCE}. Thus, all objects listed in the sequence are defined as an integral part of the table. Tables are defined as either mandatory or optional. If a table is mandatory, all of the objects with STATUS "mandatory" shall be present. If a table is optional, all of the objects with the STATUS "mandatory" shall be present if the device supports the table. Objects in the table with the STATUS "optional" may be supported.

[1] *Copyright release of Annex B Information Profile:* Users of this standard may freely reproduce this Annex so that it may be used for its intended purpose and may further publish the completed Information Profile.

B.1 NOTATION

The following notations and symbols are used to indicate status and conditional status within this standard.

B.1.1 TYPE Symbols

The following symbols are used to indicate type:

Symbol	Type
C	Control Object - use of 'dbCreateTransaction' in NTCIP 1201 Clause 2.3.1 shall NOT delay a SET to this object.
P	Parameter Object - use of 'dbCreateTransaction' in NTCIP 1201 Clause 2.3.1 to SET this object is optional.
P2	Parameter Object - use of 'dbCreateTransaction' in NTCIP 1201 Clause 2.3.1 to SET this object is mandatory.
S	Status / Information Object - this object is read only therefore a SET is not permitted.

B.1.2 Status Symbols

The following symbols are used to indicate status:

Symbol	Status
M	Mandatory
M.<n>	Support of every item of the group labeled by the same numeral <n> required, but only one is active at time.
O	Optional
O.<n>	Optional, but support of at least one of the group of options labeled by the same numeral <n> is required
C	Conditional
D	Deprecated
N/A	Non-applicable (i.e., logically impossible in the scope of the profile)
X	Excluded or prohibited

B.1.3 Conditional Status Notation

The following predicate notation is used:

Notation	Status
<predicate>: M	Item is conditional on the <predicate>.

The <predicate>: notation means that the Status following it applies only when the feature or features identified by the predicate are supported. In the simplest case, <predicate> is the identifying tag of a single item.

B.1.4 Support Column

This section is in the form of a PICS and, therefore, includes a support column. An implementer claims support of an item by circling the appropriate answer (Yes or No) in the support column:

B.2 CCTV CAMERA CONTROL REQUIREMENTS

The Conformance Group definitions for CCTV Camera Control devices are defined in this clause. A CCTV Switch has multiple functions; thus, Conformance Groups are defined for each function.

The following table lists functional requirements for a CCTV Camera Control device, and asks if the listed features have been implemented.

Ref	Areas	Clause of Profile	Status	Support
B.3	CCTV Configuration Conformance Group	NTCIP 1205 – 3.2, 3.3 and 3.11	M	YES
B.4	CCTV Extended Functions Conformance Group	NTCIP 1205 – 3.6, 3.7, 3.8, 3.9 and 3.10	O	YES
B.5	CCTV Motion Control Conformance Group	NTCIP 1205 – 3.4 and 3.5	O	YES
B.6	CCTV On-Screen Menu Control	NTCIP 1205 – 3.12	O	YES
B.7	Configuration Conformance Group	NTCIP 1201 v01, Amendment 1	M	Yes
B.8	NTCIP Security Conformance Group	NTCIP 1201 v01, Amendment 1	M	Yes

CCTV Camera Control devices shall adhere to the conformance requirements specified in the above table as a minimum to claim compliance to this standard. Additional objects or groups may be supported without being non-compliant with CCTV Camera Control objects or NTCIP.

Minimum and maximum ranges of objects that differ from the values of the object's SYNTAX field may be enforced by an application running on a device.

A device which enforces range limits within the bounds specified by the values of the object's SYNTAX field shall not be categorized as being non-compliant with CCTV Camera Control objects or NTCIP.

A device which supports a subset of objects with enumerated values shall not be categorized as being non-compliant with CCTV Camera Control objects or NTCIP.

B.3 CCTV CONFIGURATION CONFORMANCE GROUP

The CCTV Configuration Conformance Group consists of the following objects:

CCTV Configuration CONFORMANCE GROUP						
NTCIP 1205 Amend 1 Clause	**Object Name**	**Object Type**	**Object Status**	**Object Support**	**Allowed Values**	**Supported Values**
3.2, 3.3 and 3.11	**CCTV Configuration Conformance Group**	---	M	YES	----	-----
3.2	**CCTV Range Objects**	---	---	---	---	---
3.2.1	rangeMaximumPreset	S	3.2 : M	YES	0-255	64
3.2.2	rangePanLeftLimit	S	3.2 : M	YES	0-35999 | 65535	35999
3.2.3	rangePanRightLimit	S	3.2 : M	YES	0-35999 | 65535	35999
3.2.4	rangePanHomePosition	S	3.2 : M	YES	0-35999 | 65535	0
3.2.5	rangeTrueNorthOffset	P	3.2 : M	YES	0-35999 | 65535	PASSED: 0,1,18000, 35998,35999,36000, FAILED: -1,
3.2.6	rangeTiltUpLimit	S	3.2 : M	YES	0-35999 | 65535	1500
3.2.7	rangeTiltDownLimit	S	3.2 : M	YES	0-35999 | 65535	27000
3.2.8	rangeZoomLimit	S	3.2 : M	YES	0-65535	65535
3.2.9	rangeFocusLimit	S	3.2 : M	YES	0-65535	0
3.2.10	rangeIrisLimit	S	3.2 : M	YES	0-65535	0
3.2.11	rangeMinimumPanStepAngle	S	3.2 : M	YES	0-35999 | 65535	10
3.2.12	rangeMinimumTiltStepAngle	S	3.2 : M	YES	0-35999 | 65535	10
3.3	**CCTV Timeout Objects**	---	---	---	---	---
3.3.1	timeoutPan	P	3.3 : M	YES	0-65535	PASSED: 1, FAILED: -1,0,32767, 65535,65536,
3.3.2	timeoutTilt	P	3.3 : M	YES	0-65535	PASSED: 1, FAILED: -1,0,32767, 65535,65536,
3.3.3	timeoutZoom	P	3.3 : M	YES	0-65535	PASSED: 1, FAILED: -1,0,32767, 65535,65536,
3.3.4	timeoutFocus	P	3.3 : M	YES	0-65535	PASSED: 1, FAILED: -1,0,32767, 65535,65536,
3.3.5	timeoutIris	P	3.3 : M	YES	0-65535	PASSED: 1, FAILED: -1,0,32767, 65535,655

CCTV Configuration CONFORMANCE GROUP						
NTCIP 1205 Amend 1 Clause	**Object Name**	**Object Type**	**Object Status**	**Object Support**	**Allowed Values**	**Supported Values**
						36,
3.11	**CCTV Label Objects**	---	---	---	---	---
3.11.1	labelMaximum	S	3.11 : M	YES	0-255	80
3.11.2	labelTable	---	3.11 : M	YES	---	---
	labelEntry	---	3.11 : M	YES	---	---
3.11.2.1	labelIndex	S	3.11 : M	YES	0-255	
3.11.2.2	labelText	P	3.11 : M	YES	String	PASSED: "","0123456789ABCDEFGHIJ","KLMNOPQRSTUVWXYZ:./"",
3.11.2.3	***labelFontType (Deprecated in Amend v1.08)***	P	3.11 : D	YES	0-255	
3.11.2.4	labelHeight	P	3.11 : M	YES	0-255	PASSED: 256, FAILED: 0,1,127,255,
3.11.2.5	labelColor	P	3.11 : M	YES	1-16	PASSED: 0,7,17,
	blue(1)	---	---	Yes / No	---	---
	green(2)	---	---	Yes / No	---	---
	cyan(3)	---	---	Yes / No	---	---
	red(4)	---	---	Yes / No	---	---
	magenta(5)	---	---	Yes / No	---	---
	brown(6)	---	---	Yes / No	---	---
	white(7)	---	---	YES	---	---
	grey(8)	---	---	Yes / No	---	---
	lightBlue(9)	---	---	Yes / No	---	---
	lightGreen(10)	---	---	Yes / No	---	---
	lightCyan(11)	---	---	Yes / No	---	---
	lightRed(12)	---	---	Yes / No	---	---
	lightMagenta(13)	---	---	Yes / No	---	---
	yellow(14)	---	---	Yes / No	---	---
	brightWhite(15)	---	---	Yes / No	---	---
	black(16)	---	---	Yes / No	---	---
3.11.2.6	labelStartRow	P	3.11 : M	YES	0-255	PASSED: -1,1,256, FAILED: 0,127,254,255,
3.11.2.7	labelStartColumn	P	3.11 : M	YES	0-255	PASSED: -1,1,256, FAILED: 0,127,254,255,
3.11.2.8	labelStatus	S	3.11 : M	YES	String	
	bit 7 – Label is Valid for Display	---	---	Yes	---	---
	bit 6 – Display Status of Label	---	---	Yes	---	---
	bit 5 – Reserved	---	---	---	---	---
	bit 4 – Reserved	---	---	---	---	---
	bit 3 – Reserved	---	---	---	---	---
	bit 2 – Reserved	---	---	---	---	---
	bit 1 – Reserved	---	---	---	---	---
	bit 0 – Reserved	---	---	---	---	---
3.11.2.9	***labelActive (Added in Amend v1.08)***	P	3.11 : M	YES	String	PASSED: 0x00,0x80,0x7F,0x0000,
	bit 7 – Display Label	---	---	Yes	---	---
	bit 6 – Reserved	---	---	---	---	---

CCTV Configuration CONFORMANCE GROUP						
NTCIP 1205 Amend 1 Clause	**Object Name**	**Object Type**	**Object Status**	**Object Support**	**Allowed Values**	**Supported Values**
	bit 5 – Reserved	---	---	---	---	---
	bit 4 – Reserved	---	---	---	---	---
	bit 3 – Reserved	---	---	---	---	---
	bit 2 – Reserved	---	---	---	---	---
	bit 1 – Reserved	---	---	---	---	---
	bit 0 – Reserved	---	---	---	---	---
3.11.2.9	***labelFontNumber (Added in Amend v1.08)***	P	3.11 : M	YES	1-255	PASSED: 0,1,2,3,254 ,255,256,
3.11.3	labelLocationLabel	P	3.11 : M	YES	0-255	PASSED: -1,0,1,2,255 , FAILED: 256,
3.11.4	labelEnableTextDisplay	P	3.11 : M	YES	String	PASSED: 0x00,0x80, 0x0000, FAILED: 0x7F,
	bit 7 – Display All Labels at Once	---	---	YES	---	---
	bit 6 – Reserved	---	---	---	---	---
	bit 5 – Reserved	---	---	---	---	---
	bit 4 – Reserved	---	---	---	---	---
	bit 3 – Reserved	---	---	---	---	---
	bit 2 – Reserved	---	---	---	---	---
	bit 1 – Reserved	---	---	---	---	---
	bit 0 – Reserved	---	---	---	---	---

B.4 CCTV EXTENDED FUNCTIONS CONFORMANCE GROUP

The CCTV Extended Functions Conformance Group consists of the following objects:

CCTV Extended Functions CONFORMANCE GROUP						
NTCIP 1205 Amend 1 Clause	Object Name	Object Type	Object Status	Object Support	Allowed Values	Supported Values
3.6, 3.7, 3.8, 3.9 and 3.10	**CCTV Extended Functions Conformance Group**	---	O	YES	---	---
3.6	**CCTV System Feature Control Objects**	---	---	---	---	---
3.6.1	systemCameraFeatureControl	C	3.6 : M	YES	String	PASSED: 0x8000,0x0080, FAILED: 0x4000,0x2000,0x1000,0x0800, 0x037F,
	Byte 1, bit 7 – Camera Power Select	---	---	Yes	---	---
	Byte 1, bit 6 – Heater Power Select	---	---	Yes	---	---
	Byte 1, bit 5 – Wiper Select	---	---	Yes	---	---
	Byte 1, bit 4 – Washer Select	---	---	Yes	---	---
	Byte 1, bit 3 – Blower Select	---	---	Yes	---	---
	Byte 1, bit 2 – Reserved	---	---	---	---	---
	Byte 1, bit 1 – Reserved	---	---	---	---	---
	Byte 1, bit 0 – Reserved	---	---	---	---	---
	Byte 2, bit 7 – Activation and Deactivation of the Camera Component	---	---	Yes	---	---
	Byte 2, bit 6 – Reserved	---	---	---	---	---
	Byte 2, bit 5 – Reserved	---	---	---	---	---
	Byte 2, bit 4 – Reserved	---	---	---	---	---
	Byte 2, bit 3 – Reserved	---	---	---	---	---
	Byte 2, bit 2 – Reserved	---	---	---	---	---
	Byte 2, bit 1 – Reserved	---	---	---	---	---
	Byte 2, bit 0 – Reserved	---	---	---	---	---
3.6.2	systemCameraFeatureStatus	S	3.6 : M	YES	String	
	bit 7 – Camera Power Status	---	---	Yes	---	---
	bit 6 – Heater Power Status	---	---	Yes	---	---
	bit 5 – Wiper Status	---	---	Yes	---	---
	bit 4 – Washer Status	---	---	Yes	---	---
	bit 3 – Blower Status	---	---	Yes	---	---
	bit 2 – Reserved	---	---	---	---	---
	bit 1 – Reserved	---	---	---	---	---
	bit 0 – Reserved	---	---	---	---	---
3.6.3	systemCameraEquipped	S	3.6 : M	YES	String	0x80
	bit 7 – Camera Power Available	---	---	Yes	---	---
	bit 6 – Heater Power Available	---	---	Yes	---	---
	bit 5 – Wiper Available	---	---	Yes	---	---
	bit 4 – Washer Available	---	---	Yes	---	---
	bit 3 – Blower Available	---	---	Yes	---	---
	bit 2 – Reserved	---	---	---	---	---
	bit 1 – Reserved	---	---	---	---	---
	bit 0 – Reserved	---	---	---	---	---
3.6.4	systemLensFeatureControl	C	3.6 : M	YES	String	PASSED: 0x0000,0x8000,0x4000,0x0080, FAILED: 0x3F7F,
	Byte 1, bit 7 – Auto Iris Select	---	---	Yes	---	---
	Byte 1, bit 6 – Auto Focus Select	---	---	Yes	---	---
	Byte 1, bit 5 – Reserved	---	---	---	---	---
	Byte 1, bit 4 – Reserved	---	---	---	---	---
	Byte 1, bit 3 – Reserved	---	---	---	---	---
	Byte 1, bit 2 – Reserved	---	---	---	---	---

CCTV Extended Functions CONFORMANCE GROUP						
NTCIP 1205 Amend 1 Clause	**Object Name**	**Object Type**	**Object Status**	**Object Support**	**Allowed Values**	**Supported Values**
	Byte 1, bit 1 – Reserved	---	---	---	---	---
	Byte 1, bit 0 – Reserved	---	---	---	---	---
	Byte 2, bit 7 – Activation and Deactivation of the Lens Component	---	---	Yes	---	---
	Byte 2, bit 6 – Reserved	---	---	---	---	---
	Byte 2, bit 5 – Reserved	---	---	---	---	---
	Byte 2, bit 4 – Reserved	---	---	---	---	---
	Byte 2, bit 3 – Reserved	---	---	---	---	---
	Byte 2, bit 2 – Reserved	---	---	---	---	---
	Byte 2, bit 1 – Reserved	---	---	---	---	---
	Byte 2, bit 0 – Reserved	---	---	---	---	---
3.6.5	systemLensFeatureStatus	S	3.6 : M	YES	String	
	bit 7 – Auto Iris Status	---	---	Yes	---	---
	bit 6 – Auto Focus Status	---	---	Yes	---	---
	bit 5 – Reserved	---	---	---	---	---
	bit 4 – Reserved	---	---	---	---	---
	bit 3 – Reserved	---	---	---	---	---
	bit 2 – Reserved	---	---	---	---	---
	bit 1 – Reserved	---	---	---	---	---
	bit 0 – Reserved	---	---	---	---	---
3.6.6	systemLensEquipped	S	3.6 : M	YES	String	0xc0
	bit 7 – Auto Iris Available	---	---	Yes	---	---
	bit 6 – Auto Focus Available	---	---	Yes	---	---
	bit 5 – Reserved	---	---	---	---	---
	bit 4 – Reserved	---	---	---	---	---
	bit 3 – Reserved	---	---	---	---	---
	bit 2 – Reserved	---	---	---	---	---
	bit 1 – Reserved	---	---	---	---	---
	bit 0 – Reserved	---	---	---	---	---
3.7	**CCTV Alarm Objects**	---	---	---	---	---
3.7.1	alarmStatus	S	3.7 : M	YES	String	
	bit 7 – Cabinet Alarm Status	---	---	Yes	---	---
	bit 6 – Enclosure Alarm Status	---	---	Yes	---	---
	bit 5 – Video Loss Alarm Status	---	---	Yes	---	---
	bit 4 – Temperature Alarm Status	---	---	Yes	---	---
	bit 3 – Pressure Alarm Status	---	---	Yes	---	---
	bit 2 – Local/Remote Alarm Status	---	---	Yes	---	---
	bit 1 – Washer Fluid Alarm Status	---	---	Yes	---	---
	bit 0 – Reserved	---	---	---	---	---
3.7.2	alarmLatchStatus	S	3.7 : M	YES	String	
	bit 7 – Cabinet Alarm Latch Status	---	---	Yes	---	---
	bit 6 – Enclosure Alarm Latch Status	---	---	Yes	---	---
	bit 5 – Video Loss Alarm Latch Status	---	---	Yes	---	---
	bit 4 – Temperature Alarm Latch Status	---	---	Yes	---	---
	bit 3 – Pressure Alarm Latch Status	---	---	Yes	---	---
	bit 2 – Local/Remote Alarm Latch Status	---	---	Yes	---	---
	bit 1 – Washer Fluid Alarm Latch Status	---	---	Yes	---	---
	bit 0 – Reserved	---	---	---	---	---
3.7.3	alarmLatchClear	P	3.7 : M	YES	String	PASSED: 0x00,0x80, 0x40,0x20, 0x10,0x08, 0x04,0x02, FAILED: 0x01,
	bit 7 – Cabinet Alarm Latch Clear	---	---	Yes	---	---
	bit 6 – Enclosure Alarm Latch Clear	---	---	Yes	---	---
	bit 5 – Video Loss Alarm Latch Clear	---	---	Yes	---	---
	bit 4 – Temperature Alarm Latch Clear	---	---	Yes	---	---
	bit 3 – Pressure Alarm Latch Clear	---	---	Yes	---	---
	bit 2 – Local/Remote Alarm Latch Clear	---	---	Yes	---	---

CCTV Extended Functions CONFORMANCE GROUP						
NTCIP 1205 Amend 1 Clause	**Object Name**	**Object Type**	**Object Status**	**Object Support**	**Allowed Values**	**Supported Values**
	bit 1 – Washer Fluid Alarm Latch Clear	---	---	Yes	---	---
	bit 0 – Reserved	---	---	---	---	---
3.7.4	alarmTemperatureHighLowThreshold	P	3.7 : M	YES	String	PASSED: 0x0000,0x0028,0x807F,0x000000,
	Byte 1 – Low Temperature Threshold	---	---	Yes	---	---
	Byte 2 – High Temperature Threshold	---	---	Yes	---	---
3.7.5	alarmTemperatureCurrentValue	S	3.7 : M	YES	String	
	Byte 1 – Current Temperature Value	---	---	Yes	---	---
3.7.6	alarmPressurehighLowThreshold	P	3.7 : M	YES	String	PASSED: 0x0000,0x0528,0x0014,0x000000,
	Byte 1 – Low Pressure Threshold	---	---	Yes	---	---
	Byte 2 – High Pressure Threshold	---	---	Yes	---	---
3.7.7	alarmPressureCurrentValue	S	3.7 : M	YES	String	
	Byte 1 – Current Pressure Value	---	---	Yes	---	---
3.7.8	alarmWasherFluidHighLowThreshold	P	3.7 : M	YES	String	PASSED: 0x0000,0x0A5A,0x0555,0x000000,
	Byte 1 – Low Washer Fluid Threshold	---	---	Yes	---	---
	Byte 2 – High Washer Fluid Threshold	---	---	Yes	---	---
3.7.9	alarmWasherFluidCurrentValue	S	3.7 : M	YES	String	
	Byte 1 – Current Washer Fluid Value	---	---	Yes	---	---
3.7.10	alarmLabelIndex	P		YES		FAILED: 0x00000000000000,0x01020304050607,0x07060504030201,0xFFFFFFFFFFFFFF,0xFFFFFFFFFFFFFF,
	Byte 1 – Cabinet Alarm Label Number	---	---	Yes	---	---
	Byte 2 – Enclosure Alarm Label Number	---	---	Yes	---	---
	Byte 3 – Video Loss Alarm Label Number	---	---	Yes	---	---
	Byte 4 – Temperature Alarm Label Number	---	---	Yes	---	---
	Byte 5 – Pressure Alarm Label Number	---	---	Yes	---	---
	Byte 6 – Local/Remote Alarm Label Number	---	---	Yes	---	---
	Byte 7 – Washer Fluid Alarm Label Number	---	---	Yes	---	---
3.7.11	***alarmLabelSource (Added in Amend v1.08)***	P	3.7 : M	YES	String	PASSED: 0x00,0x54,0xFF,0x0000,
	bit 7 – Cabinet Alarm Latch Status	---	---	Yes	---	---
	bit 6 – Enclosure Alarm Latch Status	---	---	Yes	---	---
	bit 5 – Video Loss Alarm Latch Status	---	---	Yes	---	---
	bit 4 – Temperature Alarm Latch Status	---	---	Yes	---	---
	bit 3 – Pressure Alarm Latch Status	---	---	Yes	---	---
	bit 2 – Local/Remote Alarm Latch Status	---	---	Yes	---	---
	bit 1 – Washer Fluid Alarm Latch Status	---	---	Yes	---	---
	bit 0 – Reserved	---	---	---	---	---
3.8	**CCTV Discrete Input Objects**	---	---	---	---	---
3.8.1	inputStatus	S	3.8 : M	YES	String	
	bit 7 – Discrete Input 8 Active Status	---	---	Yes	---	---

CCTV Extended Functions CONFORMANCE GROUP						
NTCIP 1205 Amend 1 Clause	Object Name	Object Type	Object Status	Object Support	Allowed Values	Supported Values
	bit 6 – Discrete Input 7 Active Status	---	---	Yes	---	---
	bit 5 – Discrete Input 6 Active Status	---	---	Yes	---	---
	bit 4 – Discrete Input 5 Active Status	---	---	Yes	---	---
	bit 3 – Discrete Input 4 Active Status	---	---	Yes	---	---
	bit 2 – Discrete Input 3 Active Status	---	---	Yes	---	---
	bit 1 – Discrete Input 2 Active Status	---	---	Yes	---	---
	bit 0 – Discrete Input 1 Active Status	---	---	Yes	---	---
3.8.2	inputLatchStatus	S	3.8 : M	YES	String	
	bit 7 – Discrete Input 8 Latch Status	---	---	Yes	---	---
	bit 6 – Discrete Input 7 Latch Status	---	---	Yes	---	---
	bit 5 – Discrete Input 6 Latch Status	---	---	Yes	---	---
	bit 4 – Discrete Input 5 Latch Status	---	---	Yes	---	---
	bit 3 – Discrete Input 4 Latch Status	---	---	Yes	---	---
	bit 2 – Discrete Input 3 Latch Status	---	---	Yes	---	---
	bit 1 – Discrete Input 2 Latch Status	---	---	Yes	---	---
	bit 0 – Discrete Input 1 Latch Status	---	---	Yes	---	---
3.8.3	inputLatchClear	C	3.8 : M	YES	String	PASSED: 0x00,0xFF, 0x55,0x0000,
	bit 7 – Discrete Input 8 Latch Clear	---	---	Yes	---	---
	bit 6 – Discrete Input 7 Latch Clear	---	---	Yes	---	---
	bit 5 – Discrete Input 6 Latch Clear	---	---	Yes	---	---
	bit 4 – Discrete Input 5 Latch Clear	---	---	Yes	---	---
	bit 3 – Discrete Input 4 Latch Clear	---	---	Yes	---	---
	bit 2 – Discrete Input 3 Latch Clear	---	---	Yes	---	---
	bit 1 – Discrete Input 2 Latch Clear	---	---	Yes	---	---
	bit 0 – Discrete Input 1 Latch Clear	---	---	Yes	---	---
3.8.4	inputLabelIndex	P	3.8 : M	YES	String	FAILED: 0x000000, 0x010203, 0xFFFFFF, 0x01020304,
	Byte 1 – Discrete Input 1 Label Number	---	---	Yes	---	---
	Byte 2 – Discrete Input 2 Label Number	---	---	Yes	---	---
	Byte 3 – Discrete Input 3 Label Number	---	---	Yes	---	---
	Byte 4 – Discrete Input 4 Label Number	---	---	Yes	---	---
	Byte 5 – Discrete Input 5 Label Number	---	---	Yes	---	---
	Byte 6 – Discrete Input 6 Label Number	---	---	Yes	---	---
	Byte 7 – Discrete Input 7 Label Number	---	---	Yes	---	---
	Byte 8 – Discrete Input 8 Label Number	---	---	Yes	---	---
3.8.5	***inputPresetIndex (Added in Amend v1.08)***	P	3.8 : M	YES	String	PASSED: 0x0000000000000000,0x0102030405060708,0x0807060504030201,0xFFFFFFFFFFFFFFFF,
	Byte 1 – Discrete Input 1 Preset Number	---	---	Yes	---	---
	Byte 2 – Discrete Input 2 Preset Number	---	---	Yes	---	---
	Byte 3 – Discrete Input 3 Preset Number	---	---	Yes	---	---
	Byte 4 – Discrete Input 4 Preset Number	---	---	Yes	---	---
	Byte 5 – Discrete Input 5 Preset Number	---	---	Yes	---	---
	Byte 6 – Discrete Input 6 Preset Number	---	---	Yes	---	---
	Byte 7 – Discrete Input 7 Preset Number	---	---	Yes	---	---
	Byte 8 – Discrete Input 8 Preset Number	---	---	Yes	---	---
3.8.6	***inputLabelSource (Added in Amend v1.08)***	P	3.8 : M	YES	String	PASSED: 0x00,0xFF,

CCTV Extended Functions CONFORMANCE GROUP						
NTCIP 1205 Amend 1 Clause	Object Name	Object Type	Object Status	Object Support	Allowed Values	Supported Values
						0x0000,
	bit 7 – Discrete Input 8 Label Source	---	---	Yes	---	---
	bit 6 – Discrete Input 7 Label Source	---	---	Yes	---	---
	bit 5 – Discrete Input 6 Label Source	---	---	Yes	---	---
	bit 4 – Discrete Input 5 Label Source	---	---	Yes	---	---
	bit 3 – Discrete Input 4 Label Source	---	---	Yes	---	---
	bit 2 – Discrete Input 3 Label Source	---	---	Yes	---	---
	bit 1 – Discrete Input 2 Label Source	---	---	Yes	---	---
	bit 0 – Discrete Input 1 Label Source	---	---	Yes	---	---
3.9	**CCTV Discrete Output Objects**	---	---	---	---	---
3.9.1	outputStatus	S	3.9 : M	YES	String	
	bit 7 – Discrete Output 8 Active Status	---	---	Yes	---	---
	bit 6 – Discrete Output 7 Active Status	---	---	Yes	---	---
	bit 5 – Discrete Output 6 Active Status	---	---	Yes	---	---
	bit 4 – Discrete Output 5 Active Status	---	---	Yes	---	---
	bit 3 – Discrete Output 4 Active Status	---	---	Yes	---	---
	bit 2 – Discrete Output 3 Active Status	---	---	Yes	---	---
	bit 1 – Discrete Output 2 Active Status	---	---	Yes	---	---
	bit 0 – Discrete Output 1 Active Status	---	---	Yes	---	---
3.9.2	outputControl	C	3.9 : M	YES	String	PASSED: 0x0000,0xFFFF,0x000000,
	Byte 1, bit 7 – Discrete Output 8 Control	---	---	Yes	---	---
	Byte 1, bit 6 – Discrete Output 7 Control	---	---	Yes	---	---
	Byte 1, bit 5 – Discrete Output 6 Control	---	---	Yes	---	---
	Byte 1, bit 4 – Discrete Output 5 Control	---	---	Yes	---	---
	Byte 1, bit 3 – Discrete Output 4 Control	---	---	Yes	---	---
	Byte 1, bit 2 – Discrete Output 3 Control	---	---	Yes	---	---
	Byte 1, bit 1 – Discrete Output 2 Control	---	---	Yes	---	---
	Byte 1, bit 0 – Discrete Output 1 Control	---	---	Yes	---	---
	Byte 2, bit 7 – Discrete Output 8 Active	---	---	Yes	---	---
	Byte 2, bit 6 – Discrete Output 7 Active	---	---	Yes	---	---
	Byte 2, bit 5 – Discrete Output 6 Active	---	---	Yes	---	---
	Byte 2, bit 4 – Discrete Output 5 Active	---	---	Yes	---	---
	Byte 2, bit 3 – Discrete Output 4 Active	---	---	Yes	---	---
	Byte 2, bit 2 – Discrete Output 3 Active	---	---	Yes	---	---
	Byte 2, bit 1 – Discrete Output 2 Active	---	---	Yes	---	---
	Byte 2, bit 0 – Discrete Output 1 Active	---	---	Yes	---	---
3.9.3	outputLabelIndex	P	3.9 : M	YES	String	FAILED: 0x0000000000000000,0x0102030405060708,0x0807060504030201,0xFFFFFFFFFFFFFFFF,0xFFFFFFFFFFFFFFFF,
	Byte 1 – Discrete Output 1 Label Number	---	---	Yes	---	---
	Byte 2 – Discrete Output 2 Label Number	---	---	Yes	---	---
	Byte 3 – Discrete Output 3 Label Number	---	---	Yes	---	---
	Byte 4 – Discrete Output 4 Label Number	---	---	Yes	---	---
	Byte 5 – Discrete Output 5 Label Number	---	---	Yes	---	---
	Byte 6 – Discrete Output 6 Label Number	---	---	Yes	---	---
	Byte 7 – Discrete Output 7 Label Number	---	---	Yes	---	---
	Byte 8 – Discrete Output 8 Label Number	---	---	Yes	---	---
3.10	**CCTV Zone Objects**	---	---	---	---	---

CCTV Extended Functions CONFORMANCE GROUP						
NTCIP 1205 Amend 1 Clause	**Object Name**	**Object Type**	**Object Status**	**Object Support**	**Allowed Values**	**Supported Values**
3.10.1	zoneMaximum	S	3.10 : M	YES	0-255	16
3.10.2	zoneTable	---	3.10 : M	YES	---	---
	zoneEntry	---	3.10 : M	YES	---	---
3.10.2.1	zoneIndex	S	3.10 : M	YES	0-255	
3.10.2.2	zoneLabel	P	3.10 : M	YES	0-255	PASSED: 0,8,255, FAILED: -1,256,
3.10.2.3	zonePanLeftLimit	P	3.10 : M	YES	1-35999 | 65535	PASSED: 0,35999,65536, FAILED: -1,35600,65535,
3.10.2.4	zonePanRightLimit	P	3.10 : M	YES	1-35999 | 65535	PASSED: 0,35999,36000,65535, FAILED: -1,65536,
3.10.2.5	zoneTiltUpLimit	P	3.10 : M	YES	1-35999 | 65535	PASSED: -1,36000,65536, FAILED: 0,35999,65535,
3.10.2.6	zoneTiltDownLimit	P	3.10 : M	YES	1-35999 | 65535	PASSED: -1,36000,65536, FAILED: 0,35999,65535,
3.10.2.7	***zoneVideoControl (Added in Amend v1.08)***	C	3.10 : M	YES	String	FAILED: 0x00,0x80,0x7F,0x0000,
	bit 7 – Video Signal Output Control	---	---	Yes	---	---
	bit 6 – Reserved	---	---	---	---	---
	bit 5 – Reserved	---	---	---	---	---
	bit 4 – Reserved	---	---	---	---	---
	bit 3 – Reserved	---	---	---	---	---
	bit 2 – Reserved	---	---	---	---	---
	bit 1 – Reserved	---	---	---	---	---
	bit 0 – Reserved	---	---	---	---	---
3.10.3	***zoneCameraEquipped (Added in Amend v1.08)***	S	3.10 : M	YES	String	0xe0
	bit 7 – Zones Availability	---	---	Yes	---	---
	bit 6 – Zone Labels Availability	---	---	Yes	---	---
	bit 5 – Video Signal Control Availability	---	---	Yes	---	---
	bit 4 – Reserved	---	---	---	---	---
	bit 3 – Reserved	---	---	---	---	---
	bit 2 – Reserved	---	---	---	---	---
	bit 1 – Reserved	---	---	---	---	---
	bit 0 – Reserved	---	---	---	---	---

B.5 CCTV MOTION CONTROL CONFORMANCE GROUP

The CCTV Motion Control Conformance Group shall consist of the following objects:

CCTV Motion Control CONFORMANCE GROUP

NTCIP 1205 Amend 1 Clause	Object Name	Object Type	Object Status	Object Support	Allowed Values	Supported Values
3.4 and 3.5	**CCTV Motion Control Conformance Group**	---	O	YES	----	-----
3.4	**CCTV Preset Objects**	---	---	---	---	---
3.4.1	presetGotoPosition	C	3.4 : M	YES	1-255	PASSED: 1,10, FAILED: -1,0,256,
3.4.2	presetStorePosition	P	3.4 : M	YES	1-255	PASSED: 1,10, FAILED: -1,0,256,
3.4.3	***presetPositionQuery (Added in Amend v1.08)***	S	3.4 : M	YES	0-255	
3.5	**CCTV Positioning Objects**	---	---	---	---	---
3.5.1	positionPan	C	3.5 : M	YES	String	PASSED: 0x0000000000,0x00000000,0x00000001,0x00008C9F,0x033F0000,0x037F0000,0x03BF0000,0x03FF0000,0x01013A98,0x017F3A98,0x01B13A98,0x01FF3A98,0x02013A98,0x027F3A98,0x02B13A98,0x02FF3A98,0x04000000,
3.5.2	positionTilt	C	3.5 : M	YES	String	PASSED: 0x0000000000,0x00000000,0x00000001,0x00008C9F,0x033F0000,0x037F0000,0x03BF0000,0x03FF0000,0x01013A98,0x017F3A98,0x01B13A98,0x01FF3A98,0x02013A98,0x027F3A98,0x02B13A98,0x02FF3A98,0x04000000,
3.5.3	positionZoomLens	C	3.5 : M	YES	String	PASSED: 0x0000000000,0x00000000,0x00000001,0x00008C9F,0x033F0

CCTV Motion Control CONFORMANCE GROUP						
NTCIP 1205 Amend 1 Clause	**Object Name**	**Object Type**	**Object Status**	**Object Support**	**Allowed Values**	**Supported Values**
						000,0x037F0000,0x03BF0000,0x03FF0000,0x01013A98,0x017F3A98,0x01B13A98,0x01FF3A98,0x02013A98,0x027F3A98,0x02B13A98,0x02FF3A98,0x04000000,
3.5.4	positionFocusLens	C	3.5 : M	YES	String	PASSED: 0x0000000000,0x00000000,0x00000001,0x00008C9F,0x033F0000,0x037F0000,0x03BF0000,0x03FF0000,0x04000000, FAILED: 0x01013A98,0x017F3A98,0x01B13A98,0x01FF3A98,0x02013A98,0x027F3A98,0x02B13A98,0x02FF3A98,
3.5.5	positionIrisLens	C	3.5 : M	YES	String	PASSED: 0x0000000000,0x00000000,0x00000001,0x00008C9F,0x033F0000,0x037F0000,0x03BF0000,0x03FF0000,0x04000000, FAILED: 0x01013A98,0x017F3A98,0x01B13A98,0x01FF3A98,0x02013A98,0x027F3A98,0x02B13A98,0x02FF3A98,

CCTV Motion Control CONFORMANCE GROUP						
NTCIP 1205 Amend 1 Clause	**Object Name**	**Object Type**	**Object Status**	**Object Support**	**Allowed Values**	**Supported Values**
3.5.6	***positionQueryPan (Added in Amend v1.08)***	S	3.5 : M	YES	1-35999 | 65535	
3.5.7	***positionQueryTilt (Added in Amend v1.08)***	S	3.5 : M	YES	1-35999 | 65535	
3.5.8	***positionQueryZoom (Added in Amend v1.08)***	S	3.5 : M	YES	1-65535	
3.5.9	***positionQueryFocus (Added in Amend v1.08)***	S	3.5 : O	YES	1-65535	
3.5.10	***positionQueryIris (Added in Amend v1.08)***	S	3.5 : O	YES	1-65535	

B.6 CCTV ON-SCREEN MENU CONTROL CONFORMANCE GROUP

The CCTV On-Screen Menu Control Conformance Group shall consist of the following objects:

CCTV On-Screen Menu Control CONFORMANCE GROUP						
NTCIP 1205 Amend 1 Clause	**Object Name**	**Object Type**	**Object Status**	**Object Support**	**Allowed Values**	**Supported Values**
3.12	**CCTV On-Screen Menu Control Conformance Group**	---	O	YES	----	-----
3.12	**CCTV On-Screen Camera Menu Objects**	---	---	---	---	---
3.12.1	menuActivate	P	3.12 : M	YES	0-255	PASSED: 0,1,254,255, FAILED: -1,256,
3.12.2	menuControl	C	3.12 : M	YES	1-255	
	pageDown(1)	---	---	Yes / No	---	---
	pageUp(2)	---	---	Yes / No	---	---
	cursorUp(3)	---	---	Yes / No	---	---
	cursorDown(4)	---	---	Yes / No	---	---
	cursorRight(5)	---	---	Yes / No	---	---
	incrementValue(7)	---	---	Yes / No	---	---
	decrementValue(8)	---	---	Yes / No	---	---
	enterValue(9)	---	---	Yes / No	---	---
	noMenu(255)	---	---	Yes / No	---	---

B.7 GLOBAL CONFIGURATION CONFORMANCE GROUP

The Global Configuration Conformance Group shall consist of the following objects:

Global Configuration CONFORMANCE GROUP						
NTCIP 1201 Clause	**Object Name**	**Object Type**	**Object Status**	**Object Support**	**Allowed Values**	**Supported Values**
2.2	**Global Config Objects**	---	M	Yes	---	---
2.2.1	globalSetIDParmeter	S	2.2 : O	Yes / No	0-65535	
2.2.2	globalMaxModules	S	2.2 : M	Yes	0-255	
2.2.3	globalModuleTable	---	2.2 : M	Yes	---	---
	moduleTableEntry	---	2.2 : M	Yes	---	---
2.2.3.1	moduleNumber	S	2.2 : M	Yes	1-255	
2.2.3.2	moduleDeviceNode	S	2.2 : M	Yes	OID	
2.2.3.3	moduleMake	S	2.2 : M	Yes	String	
2.2.3.4	moduleModel	S	2.2 : M	Yes	String	
2.2.3.5	moduleVersion	S	2.2 : M	Yes	String	
2.2.3.6	moduleType	S	2.2 : M	Yes	1-3	
	other(1)	---	---	Yes / No	---	---
	hardware(2)	---	---	Yes / No	---	---
	software(3)	---	---	Yes / No	---	---

B.8 NTCIP SECURITY CONFORMANCE GROUP

The NTCIP Security Conformance Group shall consist of the following objects:

Security CONFORMANCE GROUP						
NTCIP 1201 Amend 1 Clause	**Object Name**	**Object Type**	**Object Status**	**Object Support**	**Allowed Values**	**Supported Values**
A.10	**Security Conformance Group**	--	M	Yes	----	---
A.10.1	adminCommunityName	C	A.10 : M	Yes	String	
A.10.2	maxCommunityNames	C	A.10 : M	Yes	1..255	
A.10.3	communityNameTable	--	A.10 : M	Yes	---	---
	communityNameTableEntry	--	A.10 : M	Yes	---	---
A.10.3.1	communityNameIndex	S	A.10 : M	Yes	1..255	
A.10.3.2	communityNameUser	S	A.10 : M	Yes	String	
A.10.3.3	communityNameAccessMask	S	A.10 : M	Yes	Gauge	

§

REFERENCES FOR APPENDIX D

1. NTCIP 1205 – Object Definitions for Closed Circuit Television (CCTV) Camera Control, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=1205. Accessed July 25, 2005.

APPENDIX E:
COMMUNICATIONS AND MISCELLANEOUS TEST PROCEDURES

INTRODUCTION

This appendix describes several sets of test procedures that may be applicable to NTCIP conformant field devices. The test procedures include procedures for checking the Simple Network Management Protocol (SNMP), the Transportation Transport Profile, the Point to Multi-Point Protocol with RS232 Profile, the Simple Transportation Management Protocol (STMP), and serial communication's data rates. The original basis of the test cases was the requirements in NTCIP 2001 – Class B Profile (*1*). The NTCIP standards group has subsequently rescinded the original standard and replaced it with several others. The documentation and summaries come from the NTCIP Laboratory Testing for Actuated Signal Controllers summary report (*2*). Test scripts implementing the test procedures are available at www.itstestlab.org.

SNMP TEST CASES

Table E-1 summarizes a set of test cases for checking implementations supporting the SNMP. Except for the SNMP Conformance Group test case, the procedures check the functional aspects of SNMP and are not object related. The test cases are generic but require support of an object with a specific syntax and constraint that may not be available in all field devices.

Table E-1. SNMP Test Case Summary.

SNMP Test Cases		
ID	**Title**	**Description**
General		
TC001	Setup	General setup for testing SNMP
TC002	MIB Walk	This test ensures that GETNEXT requests are properly functioning.
TC003	Get and Set an Object	This test ensures that GET and SET operations return a valid value and do not produce errors.
TC004	Get and Set Multiple Objects	This test ensures that multiple objects can be set correctly with SET operation.

Table E-1. SNMP Test Case Summary (continued).

SNMP Test Cases		
ID	**Title**	**Description**
Error Responses		
TC005	Error In Get of Multiple Objects (tooBig)	This test ensures that GET function returns a tooBig error when a response protocol data unit (PDU) exceeds snmp-maxPacketSize.
TC006	Error in Set (badValue)	This test ensures that a bad (Invalid) value is not set using SET function.
TC007	Error in Set (readOnly)	This test ensures that SET function cannot change value of a 'readOnly' object.
TC008	Error in Set (noSuchName)	This test ensures that a value cannot be set for a non-existing object.
TC009	Error in Setting Multiple Objects (badValue)	This test ensures that while setting multiple objects, if one object is set to a bad value then other valid values are not changed for the other objects.
Community Name		
TC010	Invalid Community Name	This test ensures that if community name is invalid then no object is returned with SET function.
Statistics Conformance Group		
TC011	SNMP Conformance Group	This procedure tests whether 2 objects in the SNMP (Statistics) Conformance Group are instantiated.
Encoding		
TC012	INTEGER Encoding	Check whether an object with "SYNTAX INTEGER" can accept an instance value of 0 when the length byte is set to 1, 2, 3, 4, and 5 bytes using the "short definite form" and when the "long definite form" is used. Also checks setting values of >2147483647.
TC013	INTEGER (0..255) Encoding	Check whether an object with "SYNTAX INTEGER (0..255)" can accept an instance value of 0 when the length byte is set to 1 and 2 bytes using the "short definite form" and when the "long definite form" is used.

Table E-1. SNMP Test Case Summary (continued).

SNMP Test Cases		
ID	**Title**	**Description**
TC014	INTEGER (0..65535) Encoding	Check whether an object with SYNTAX INTEGER (0..65535) can accept an instance value of 0 when the length byte is set to 3 bytes using both the “short definite form” and the “long definite form.”
TC015	INTEGER (0..4294967295) Encoding	Check whether an object with SYNTAX INTEGER (0..4294967295) [INTEGER Tag 02] can accept an instance value of 0 when the length byte is set to 3 bytes using both the short definite form and the long definite form.
TC016	INTEGER Wrong Tag Encoding	Check whether an object as SYNTAX INTEGER (0..255) [INTEGER Tag 02] would can accept an instance value with an incorrect Tag (OCTET STRING, OBJECT IDENTIFIER, SEQUENCE, IpAddress, Counter, Gauge, TimeTicks, Opaque).
TC017	OCTET STRING Encoding	Check whether an object with OCTET STRING [Tag 0x04] can accept an instance value of NULL [Tag 0x05], a 4 character string using the short definite length form, an 127 character string using the short definite length form, and a 2 each - 4 character strings using the long definite length form.
TC018	OCTET STRING Constrained Encoding	Check whether an object with SYNTAX OCTET STRING [Tag 0x04] and a size constraint is processed correctly.
TC019	Object Identifier and Null Encoding	Check whether an object with SYNTAX OBJECT IDENTIFIER [Tag 0x06] would accept as an instance value of NULL [Tag 0x05] and a valid length.
TC021	Counter Encoding	Check whether an object with SYNTAX Counter [Tag 0x41] is processed correctly.
TC022	Counter (0..255) Encoding	Check whether an object with SYNTAX Counter (0..255) [Tag 0x41] is processed correctly.
TC023	Counter (0..65535) Encoding	Check whether an object with SYNTAX Counter (0..65535) [Tag 0x41] is processed correctly.
TC024	Gauge Encoding	Check whether an object with SYNTAX Gauge [Tag 0x42] is processed correctly.
TC025	Gauge (0..255) Encoding	Check whether an object with SYNTAX Gauge (0..255) [Tag 0x42] is processed correctly.
TC026	Gauge (0..65535) Encoding	Check whether an object with SYNTAX Gauge (0..65535) [Tag 0x42] is processed correctly.

Table E-1. SNMP Test Case Summary (continued).

SNMP Test Cases		
ID	**Title**	**Description**
TC027	TimeTicks Encoding	Check whether an object with SYNTAX TimeTicks [Tag 0x43] is processed correctly.
Opaque Encoding		
TC028	Opaque Encoding – Setup	This procedure performs a general setup prior to executing any of the specific Opaque Encoding Test Procedures.
TC029	Opaque Encoding – INTEGER	Check whether an object with SYNTAX Opaque [Tag 0x44] that encodes an instance of INTEGER is processed correctly.
TC030	Opaque Encoding - INTEGER (0..255)	Check whether an object with SYNTAX Opaque [Tag 0x44] that encodes an instance of INTEGER (0..255) is processed correctly.
TC031	Opaque Encoding - INTEGER (0..65535)	Check whether an object with SYNTAX Opaque [Tag 0x44] that encodes an instance of INTEGER (0..65535) is processed correctly.
TC032	Opaque Encoding - INTEGER (0..4294967295)	Check whether an object with SYNTAX Opaque [Tag 0x44] that encodes an instance of INTEGER (0..4294967295) is processed correctly.
TC033	Opaque Encoding - INTEGER (Constrained)	Check whether an object with SYNTAX Opaque [Tag 0x44] that encodes an instance of INTEGER (Constrained) is processed correctly.
TC034	Opaque Encoding - OCTET STRING	Check whether an object with SYNTAX Opaque [Tag 0x44] that encodes an instance of OCTET STRING is processed correctly.
TC035	Opaque Encoding - OBJECT IDENTIFIER	Check whether an object with SYNTAX Opaque [Tag 0x44] that encodes an instance of OBJECT IDENTIFIER is processed correctly.
TC036	Opaque Encoding - IpAddress	Check whether an object with SYNTAX Opaque [Tag 0x44] that encodes an instance of IpAddress is processed correctly.
TC037	Opaque Encoding - Counter	Check whether an object with SYNTAX Opaque [Tag 0x44] that encodes an instance of Counter is processed correctly.
TC038	Opaque Encoding - Gauge	Check whether an object with SYNTAX Opaque [Tag 0x44] that encodes an instance of Gauge is processed correctly.

Table E-1. SNMP Test Case Summary (continued).

SNMP Test Cases		
ID	**Title**	**Description**
TC039	Opaque Encoding - TimeTicks	Check whether an object with SYNTAX Opaque [Tag 0x44] that encodes an instance of TimeTicks is processed correctly.
TC040	Opaque Encoding - Tear Down	This procedure performs a general restore after executing any of the specific Opaque Encoding Test Procedures.

TRANSPORTATION TRANSPORT TEST CASES

Table E-2 summarizes several test cases for checking implementations supporting the NTCIP 2201 – Transportation Transport Profile (*3*). The first two test cases test the functionality of the protocol. The Net to Media Support test case relates to an object conformance group but the group is mandatory only if an IP Address scheme is used.

Table E-2. Transportation Transport Test Case Summary.

Transportation Transport Profile		
ID	**Title**	**Description**
NULL		
TC001	Unknown IPI	This test checks whether a device under test (DUT) accepts an upper layer Protocol Data Unit with an invalid Initial Protocol Identifier.
TC002	Max Protocol Data Unit	This procedure checks whether a DUT supports the required Protocol Data Unit size.
TC003	Net to Media Support	This procedure checks for support of the ipNetToMedia conformance group.

POINT TO MULTI-POINT WITH RS232 TEST CASES

Table E-3 summarizes a set of test cases for checking implementations supporting the NTCIP 2101 – Point to Multi-Point Protocol Using RS-232 Subnetwork Profile (4). Except for the RS232 Conformance Group and the LapB Conformance Group test cases, the procedures check the functional aspects of PMPP and are not object related.

Table E-3. PMPP with RS-232 Test Case Summary.

PMPP with RS-232		
ID	**Title**	**Description**
Short Address		
TC001	Setup	General setup for PMPP Procedures
TC002	Short Address - Positive Test 1	This procedure checks whether a DUT responds to a valid short address.
TC003	Short Address - Negative Test 1	This procedure checks whether a DUT responds to another DUT's address.
TC004	Short Address - Positive Test 2	This procedure checks whether a DUT responds to another valid short address.
TC005	Short Address - Negative Test 2	This procedure checks whether a DUT responds to another DUT's short address.
TC006	Restore Default Address	This test ensures that the default address is restored properly.
Long Address		
TC007	Long Single Address - Positive Test	This procedure checks whether a DUT responds to a valid long address.
TC008	Large Single Address - Negative Test #1	This procedure checks whether a DUT responds to another DUT's long address.
TC009	Large Single Address - Negative Test #2	This procedure checks whether a DUT responds to an invalid long address.
TC010	Restore Default Address	This test ensures that the default address is restored properly.
Broadcast and Polling		
TC011	Broadcast Message	This procedure checks whether a DUT responds to a broadcast message or not.

Table E-3. PMPP with RS-232 Test Case Summary (continued).

PMPP with RS-232		
ID	**Title**	**Description**
TC012	Poll Message	This test ensures that previous broadcast message's response is buffered and then sent if Unnumbered Poll (UP) is sent to the DUT.
Group Address		
TC013	Setting Group Address	This test ensures that the group address is set properly.
TC014	Group Address - Positive Test	This procedure checks whether a DUT does not respond to a GET with a valid group address but buffers the response and returns that buffered response upon a UP.
TC015	Group Address - Negative Test	This procedure checks whether a DUT responds to a GET with an invalid group address.
TC016	Large Group Address - Positive Test	This procedure checks whether a DUT responds to a GET with a valid large group address.
TC017	Large Group Address - Negative Test	This procedure checks whether a DUT responds to a GET with an invalid large group address
Polling		
TC018	Request Without Poll Bit	This procedure checks whether a DUT responds to GET request to valid single address but without the Poll Bit set.
TC019	Poll	This procedure checks whether a DUT responds with the buffered response from the Request Without Poll Bit procedure when a UP is sent.
Control Byte		
TC020	Changed Control Byte	This procedure checks whether a DUT responds to a PDU with an invalid Control Byte.
Initial Protocol Identifier (IPI)		
TC021	Unknown IPI	This procedure checks whether a DUT responds to a PDU with an invalid IPI.

Field Check Sum

Table E-3. PMPP with RS-232 Test Case Summary (continued).

PMPP with RS-232		
ID	**Title**	**Description**
TC022	Invalid Cyclic Redundancy Check (CRC)	This procedure checks whether a DUT responds to a PDU with an invalid CRC on good data.
TC023	Changed Data	This procedure checks whether a DUT responds to a PDU with a valid CRC on bad data.
RS232 and HDLC Conformance Groups		
TC024	RS232 Conformance Group	This procedure checks whether a DUT responds to one of the objects in the RS232 Conformance Group.
TC025	HDLC Conformance Group	This procedure checks whether a DUT responds to one of the objects in the HDLC Conformance Group.
Frame Size and Buffering		
TC026	Frame Size	This procedure checks whether a DUT can accept and send a PDU that is the largest that must be supported.
TC027	Byte Stuffing	This procedure checks whether the DUT can support a PDU where 5% of the octets are byte-stuffed.

STMP TEST CASES

Table E-4 summarizes a set of test cases for checking implementations supporting the Simple Transportation Management Protocol (STMP), whose definition appears in NTCIP 1103 – Transportation Management Protocols and whose profile requirements appear in NTCIP 2301 – Simple Transportation Management Framework Application Profile *(5,6)*. The procedure defines a set of dynamic object messages that encode the various data types and then checks that the encoded data type values are valid. The documentation and test scripts for these procedures are only partially complete. However, they do provide a starting point and approach to testing the functional aspects of STMP protocol.

Table E-4. STMP Test Case Summary.

STMP Test Cases		
ID	**Title**	**Description**
TC001	General Setup	This procedure clears out or invalidates any previous dynamic object definitions.
TC002	Set Up Dynamic Objects 1 To 11 to a Single Variable	This procedure defines a set of dynamic messages consisting of a single variable.
TC003	Define Compare Values	This procedure defines a set of compare values objects for use in the single variable test.
TC004	Compare STMP results	This procedure compares the values of the single variable messages test returned using STMP with those returned using SNMP.
TC005	Set Up Dynamic 12 to 11 Variables	This procedure defines a single dynamic message consisting of 11 variables.
TC006	Compare STMP Object 12 Results	This procedure compares the values of the 11 variable dynamic messages returned using STMP with those returned using SNMP.
TC007	General Clean Up	This procedure invalidates the dynamic objects definitions.

RESPONSE TIME TEST CASE

Table E-5 summarizes a test case for checking support of the various data rates as defined in NTCIP 2101 – Point to Multi-Point Protocol Using RS-232 Subnetwork Profile and in NTCIP 2102 – Point to Multi-Point Protocol Using FSK Modem Subnetwork Profile (*4,7*). By repeating a message exchange multiple times, one can make an evaluation of the response time or length of time that an implementation takes to process a message.

Table E-5. Response Time Test Case Summary.

PMPP Data Rates and Response Times		
ID	**Title**	**Description**
TC001	Response Time	The purpose of this test procedure is to evaluate response time and test for support of various data rates defined in NTCIP 2101 and 2103. NTCIP 2101 requires 1200 bps and lists 2400, 4800, 9600, 19.2K, 38.4K, 57.6K, and others as optional. NTCIP 2102 is 1200 bps by definition. NTCIP 2103 requires support for 2400, 4800, 9600, 19.2K bps and states that higher data rates are optional.

REFERENCES FOR APPENDIX E

1. NTCIP 2001 – Class B Profile, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=2001. Accessed August 16, 2006.

2. NTCIP Laboratory Testing for Actuated Signal Controllers, Summary Report for ASSHTO Project 475070. Published by Texas Transportation Institute. http://tti.tamu.edu/documents/TTI-2006-1.pdf. Accessed June 7, 2006.

3. NTCIP 2201 – Transportation Transport Profile, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=2201. Accessed July 25, 2005.

4. NTCIP 2101– Point to Multi-Point Protocol Using RS-232 Subnetwork Profile, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=2101. Accessed June 27, 2006.

5. NTCIP 1103 – Transportation Management Protocols, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=1103. Accessed June 27, 2006.

6. NTCIP 2301– Simple Transportation Management Framework Application Profile, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=2301. Accessed June 27, 2006.

7. NTCIP 2102– Point to Multi-Point Protocol using FSK Modem Subnetwork Profile, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=2102. Accessed June 27, 2006.

APPENDIX F:
TRAFFIC SIGNAL CONTROLLER TEST DOCUMENTATION

INTRODUCTION

Since the task of developing NTCIP test procedures for one of the ITS field devices can be a significant project in and of itself, the researcher found that following some of the recommendations in the Institute of Electrical and Electronics Engineers (IEEE) Std. 829 – IEEE Standard for Software Test Documentation are useful (*1*). Prior to actually writing procedures, the IEEE standard suggests the development of an overall plan, one or more test design specifications, and test case specifications. The overall plan conveys the scope, approach, resources, and schedule of testing activities. Its primary purpose is to present a high-level view of the project to inform all interested parties. The test design specifications provide a more detailed view of the testing project. A test engineer's supervisor and any group such as a project monitoring committee uses the test design specifications to make sure that a test engineer understands the projects and is addressing what is needed. The design specification serves as part of the validation step in the project development. Test case specifications then outline individual test cases that verify specific features and functions of an implementation undergoing test. The test case specifications provide additional oversight but primarily help a test engineer organize and plan the specifics of each test case before committing to code or formal definition. These types of documents address the planning aspects of a testing project.

Full development of these documents is beyond the scope of this TxDOT research project. In the case of the CCTV test procedures, the researchers capitalized on test procedures already in the public domain, and upfront planning did not appear to be essential. For the traffic signal controllers, however, the researcher felt that it would be helpful to have a test design specification and test case specification that organizes and outlines the approach to testing the requirements that would apply to TxDOT department material specification DMS-11170, Fully Actuated, Solid-State Traffic Signal Controller Assembly (*2*). A test design specification and test case specification from a previous project were somewhat appropriate and can serve to convey the makeup of such documents (*3*).

A number of state departments of transportation are adopting International Organization for Standardization (ISO) 9000 standards in order to improve quality *(4)*. The following test design specification and test case specification are two types of document examples that would satisfy most of the ISO 9000 requirements. The following embedded specifications, initially prepared under another research project, have been modified to put them in the context of this research.

Test Design Specification

NTCIP Conformant
Traffic Signal Controller

TDS-TSC v1.02

August 31, 2006

REVISION HISTORY

Revision Date	Version Number	Description of Change
03/08/06	v1.01	Initial draft by R. De Roche
08/31/06	v1.02	Revised for inclusion in TxDOT project 0-5003

Test Design Specification

1.0 INTRODUCTION

This test design specification outlines the requirements for testing NTCIP compliant traffic signal controllers. This specification identifies the features and/or general functions to be tested. This specification also details the test approach, proposes a rationale for the definitions of the test cases, and establishes pass/fail criteria.

This test design specification defines the elements and approach to show compliance to the NTCIP related requirements of TxDOT DMS-11170, Fully Actuated, Solid-State Traffic Signal Controller Assembly (2). This document identifies the elements to test and those that will not. The test design specification also identifies the test cases and subsequent test procedures. A test case specification documents the actual values used in the testing process. The test case specification also identifies constraints on the test procedures. Test cases are separate from test designs so that test cases can apply to one or more design specifications and have application in other test plans. A test procedure identifies the specific steps involved in executing a test case. By their nature, test procedures go into detail about a systematic process. Test procedures are in a separate document so that they do not burden the other documents with extraneous detail.

To meet the first objective, tests will be designed to evaluate a DUT's conformance to the appropriate NTCIP Standards. The current versions of the standards convey requirements in the form of a Profile or Protocol Requirement List (PRL). The testing results will be primarily conveyed to participants in the Test Bed Project by recording the results on the PRL and, thus, turning it into a Profile or Protocol Implementation Conformance Specification.

To meet the second objective, tests will be designed to evaluate a DUT's compliance to additional requirements that might be imposed when a device is used in a system. These additional requirements are defined within this document and will be summarized in the form of a Device Requirements List (DRL) as defined in Annex A. The testing results will be primarily conveyed to participants in the Test Bed Project by recording the results on the DRL and, thus, turning it into a Device Implementation Conformance Specification.

This document's organization and content uses IEEE Std. 829 – IEEE Standard for Software Test Documentation as a guide (1). The purpose of the IEEE standard is to define a basic set of testing documentation that provides a common point of reference for discussion and understanding for all parties involved in the testing process. The design of the contents serves as a completeness checklist.

1.1 TEST DESIGN SPECIFICATION IDENTIFICATION

The Test Design Specification Identifier is TDS-TSC.

1.2 FEATURES TO BE TESTED

The TxDOT specification DMS-11170, Fully Actuated, Solid-State Traffic Signal Controller Assembly defines the requirements for traffic signal controllers and other components in a signal cabinet. The organization of requirements in that specification that relate specifically to controller units is:

- Hardware Design Requirements – NEMA Controller
- Time Clock
- Clock-Calendar Programming
    - Structure and Interrelationship of Programs
- Programming Requirements
    - Phase Operation

  - Pedestrian Timing
  - Coordination
- Time Base Coordination
- Diamond Operation
  - Program Requirements
  - Four-Phase Operation
  - Concurrent Timing Requirements
  - Diamond Detector Operation
  - Three-Phase Operation
- Coordination-Control Hierarchy
- Preemption
- Closed-Loop Operation and Monitoring Software
- NTCIP Compliance

This version of the test design specification focuses specifically on:
- NTCIP Compliance
- Four-Phase Operation

NOTE – Other than Diamond Four-Phase Operation, this test design specification does not cover test cases for the functional requirements of a signal controller.

The basic features and functions associated with the NTCIP Compliance requirements of DMS-11170 are:
- Object support for mandatory objects defined in:
  - Phase Conformance Group
  - Detector Conformance Group
  - Volume Occupancy Report Conformance Group
  - Unit Conformance Group
  - Special Function Conformance Group
  - Coordination Conformance Group
  - Time Base Conformance Group
    - Time Management
    - Time Base Event Schedule
  - Preempt Conformance Group
  - Ring Conformance Group
  - Channel Conformance Group
  - Overlap Conformance Group
  - TS 2 Port 1 Conformance Group
  - Configuration Conformance Group
  - Database Management Conformance Group
  - Report Conformance Group
  - PMPP Conformance Group
  - STMF Conformance Group
    - SNMP Conformance Group
    - STMP Conformance Group
  - Security Conformance Group
  - HDLC Group Address Conformance Group
- Object support for miscellaneous optional objects defined in
  - NTCIP 1201-GLO
  - NTCIP 1202-ASC
- Support Standardized Ranges
- Specific Values Defined in Minimum Project Requirements
- Null Protocol
- PMPP Protocol using an RS-232 Physical Interface
- PMPP Protocol using an FSK Modem Interface

Additional features and functions not defined within DMS-11170 but considered relevant to controller operation:

- Block Object Conformance Group
- Systems Conformance Group
- SNMP Statistics Conformance Group
- RS232 Conformance Group
- HDLC Conformance Group

1.3 FEATURES NOT TO BE TESTED

The following features and functions do not appear as requirements in DMS-11170 and appear in NTCIP 1202 as optional. They are not essential to controller operation and, therefore, will not be included:

- Auxiliary I/O Group
- SFMP Group
- Logical Name Group
- Trap Management Group
- Interfaces Group
- IP Group
- ICMP Group
- TCP Group
- UDP Group
- Ethernet Group

1.4 APPROACH REFINEMENTS

1.4.1 Conformance Group, Optional Objects, Standardized Ranges, and Project Requirements

For the following three test cases, one or more MIBs will consolidate all the required objects and eliminate non-required objects.

1.4.1.1 Conformance Group and Optional Object Support

For testing support of the conformance groups and optional objects, a test case will perform a “MIB Walk.” This MIB walk, however, will be somewhat different from what others consider to be a MIB walk. The traditional MIB walk uses SNMP Get Next operations to step through the objects that an implementation supports. The Get Next operation returns the object identifier of the next object in logical sequence. Comparing this object identifier value to what appears in a MIB can indicate what objects are present or missing. The Get Next operation also “discovers” any additional objects. While this is a good method for determining what an implementation supports and checking the Get Next operation, it does not use a MIB as the reference point or use Get operations. From the perspective of an agency purchasing equipment, the primary concern is whether equipment implements their requirements. Knowing about additional objects can be useful but the required ones are the concern. The normal operational method of retrieving object values is through the Get operation.

Given these reasons, the basis of the MIB Walk will be Get operations. Applying potential instance value extensions to the list of object names in the MIB will generate a hierarchical set of object identifiers to use in the Get operation. By stepping through each object identifier, a test procedure can determine an implementation’s object support.

NOTE – This test case will only check for the presence of objects and will not check the functionality expressed by them.

1.4.1.2 Supported Values of Instantiated Objects

For testing the supported values, a test case will rely on an external list of sample test values to use. Test cases that perform a 100% check of all the possible values are relatively easy to define. However, when one considers that a controller contains tens of thousands of objects and the typical value range of these objects is 0 to 255, the time it takes to perform a test becomes an issue. Rather than use this approach, the test case will use a sampling technique. For every read-write object that corresponds to a parameter or control, the external list will define the objects to test. The objects will also have both positive and negative test values to use on the objects.

Each object that has only a single instance will appear in the external list. At least one instance of an object that appears multiple times in a table structure should also appear in the external list. Along with the actual test value to use, there is an indication of whether the value a positive or negative test value.

NOTE – The supported values test case will only check whether an object can be set to specific values. It does not check that the written values are actually used or the controller performs the functionality expressed by a value.

1.4.1.3 Specific Values Defined in Minimum Project Requirements

The Object Range Value for Actuated Signal Controllers table in DMS-11170 refers to two types of objects: “max” objects or constants and status objects that indicate values that correspond to states of the controller. For max objects, a test case will use Get operations to retrieve the value in the implementation and then compare the value against the required value in DMS-11170. To make this test case reusable for testing equipment other than signal controllers, it will use an external list of object identifiers to determine which objects values to retrieve. Associated with each object will be the minimum project requirement value for comparison.

NOTE – The max object test case will only check for the value of the max objects. The test case will not check the functionality expressed by them. In most cases, a max object defines a number of instances. For example, the maxPhases object defines the number of phases that a controller supports. Checking the value of maxPhases does not determine whether a controller implements that number of phases.

Status objects are more difficult to validate. Status objects take on a specific value only when the controller is in a specific state. For each status object, a test case will create the conditions that produce each state. The test case will then either verify the correct state through independent means or assume some time for the controller to reach the state. The test case will then Get the value of the status object and compare to the appropriate value.

NOTE – The status object test case will test the functionality expressed by the status object either directly or indirectly.

1.4.2 SNMP Protocol

The conformance testing of the SNMP protocol will involve a number of test cases. The test cases will have the following organization:

- General
- Error Responses
- Community Name
- SNMP Statistics Conformance Group
- BER Encoding
- Opaque Encoding

1.4.3 STMP Protocol

The conformance testing of the STMP protocol will consist of defining twelve dynamic messages to retrieve object definitions that require different Octet Encoding Rules (OER). The intent is to check the encoding of the following SYNTAX types:

- INTEGER [unconstrained]
- INTEGER (0..255)
- INTEGER (0..65535)

- INTEGER (0..4394967295)
- INTEGER (-43200..43200)
- OCTET STRING [unconstrained]
- OBJECT IDENTIFIER
- IpAddress
- Counter
- Gauge
- TimeTicks

The first eleven messages will consist of a single variable corresponding to one of the syntaxes. The twelfth message will consist of eleven objects corresponding to all of the syntax types.

NOTE – The conformance tests will not address the OER encoding of the syntax Opaque.

1.4.4 Null Protocol

The conformance testing of the Null Protocol (now referred to as the Transportation Transport Profile) will involve three test cases. These will consist of:

- Unknown IPI
- Max Protocol Data Unit
- Net to Media Support

1.4.5 PMPP Protocol Using an RS-232 Physical Interface

The conformance testing of the Point-to-Point Protocol using an RS-232 will be broken down into ten areas. These areas consist of:

- Short Address
- Long Address
- Broadcast and Polling
- Group Address
- Polling
- Control Byte
- Initial Protocol Identifier
- Field Check Sum
- RS232 and HDLC Conformance Groups
- Frame Size and Buffering

1.4.6 PMPP Protocol Using an FSK Modem Interface

The conformance testing related to the use of an FSK modem will consist of using a line impairment device to simulate worst-case conditions of the transmission line and then checking for errors. The name of the test case will be:

- Bell 202T Modem Characteristics

1.4.7 Diamond Four-Phase Operation

The test cases for diamond four-phase operation will consist of two types. The first type will address sequencing, and the second type will address detector operations.

The sequencing test cases will cycle the controller through the possible sequence patterns. In that four-phase sequence there are 6 primary movements or states: 2+5, 3+5, 4+5, 1+6, 1+7, and 1+8. This equates to 15 permutations of one state transitioning to another. The first part of the sequence test will be to put the controller in one state and by means of detector calls, get it to change to another state and then back again. Table 1 lists the initial state, detector calls that are active, and the resulting sequence.

Table 1. Sequence Table with Return to Initial State

State	Calls	Sequence	Remarks
2+5	2, 3	3+5, 2+5 →	
2+5	2, 4	4+5, 2+5 →	
2+5	2, 6	5+9, 1+6, 1+13, 2+5 →	
2+5	2, 7	5+9, 1+7, 2+15, 2+5 →	
2+5	2, 8	5+9, 1+8, 2+16, 2+5 →	
3+5	3, 4	4+5, 3+5 →	
3+5	3, 6	6+11, 1+6, 1+13, 3+5 →	
3+5	3, 7	5+9, 1+7, 1+13, 3+5 →	
3+5	3, 8	5+9, 1+8, 1+13, 3+5 →	
4+5	4, 6	6+12, 1+6, 1+13, 4+5 →	
4+5	4, 7	5+9, 1+7, 1+13, 4+5 →	
4+5	4, 8	5+9, 1+8, 1+13, 4+5 →	
1+6	6, 7	1+7, 1+6 →	
1+6	6, 8	1+8, 1+6 →	
1+7	7, 8	1+8, 1+7 →	

1.4.8 Diamond Four-Phase Detector Operations

The detector operations test cases will address each detector in turn. Since most of the detectors operations involve switching, calling, and extending, test cases will create conditions that represent the states of the controller, activate the detector input, and then monitor the status of the traffic signal controller for the proper response.

1.4.9 Additional Test Cases

The following test cases from the Test Bed Project will be applied to the controller, with time permitting:

- Retrieve Log Data
- Timebase Schedule of Event
- Database Upload/Download
- System Performance Testing

1.5 TEST IDENTIFICATION

The test case specifications associated with this Test Design Specification is TCS-TSC.

1.6 FEATURE PASS FAIL CRITERIA

The Conformance Group, Optional Objects, Standardized Ranges, and Project Requirements pass-fail criteria depend upon individual test parameters. The procedures will use the NTCIP 1202-ASC Profile Requirements List (Annex A) to record results.

The other test cases will define any specific pass-fail criteria but in general, the device under test must perform the stated operation or return the expected results to “pass.”

Test Bed Project
Test Case Specifications
Actuated Signal Controller

ITL-TCS-TBP-ASC v1.10

August 31, 2006

REVISION HISTORY

Revision Date	Version Number	Description of Change
4/17/03	v1.01	Initial draft for review by C. Herrick
5/20/03	v1.02	Incorporated various comments from C. Herrick
6/2/03	v1.03	Various updates to most clauses
1/26/04	v1.04	Updated title and added document organization
4/6/04	v1.05	Changed title and ID, segmented test cases into their own sections
6/2/04	v1.06	Added filename at end and miscellaneous edits Correct Identifier
7/1/04	v1.07	Added J. Johnson's corrections
7/9/04	v1.08	Completely revised Retrieve
8/14/06	v1.09	Revised for TxDOT Project 0-5003
8/31/06	v1.10	Removed copyright

Test Case Specification

A Test Case Specification describes precisely what is to be tested. It requires identification for each test case, a description of the test items, a reference to the functions to be tested, the inputs and expected outputs, and the test case dependencies.

This organization of documents in the IEEE Std. 829 is:

- Test Plan
- Test Design Specification
- Test Case Specification
- Test Procedure Specification

1.7 TEST CASE SPECIFICATION IDENTIFIER

This Test Case Specification Identifier is: **ITL-TCS-TBP-ASC**

This Test Case Specification addresses each of the major test cases as defined in the Test Design Specification (ITL-TDS-TBP-ASC). The major test cases are:

- Object Instantiation of NTCIP 1202 and 1201
- Supported Values of Instantiated Objects
- SNMP Protocol
- Null Protocol
- PMPP using RS-232
- PMPP using FSK Modem
- System Operational Scenarios
- Retrieve Log Data - Optional Operational Scenario
- Timebase Schedule of Event and Database Upload/Download - Optional System Operational Scenarios
- System Performance Testing

Each specific test case is covered in a separate section of this document. Additional and supporting documentation appears as Annexes.

Object Instantiated Test Case

1.8 TEST ITEMS

This test case applies to devices that implement the data elements defined within NTCIP 1202 Object Definitions for Actuated Signal Controllers. While NTCIP 1202 is, technically, an information profile that defines the functional data elements related to an Actuated Signal Controller, it also enumerates other data elements that would be instantiated in a fully conformant implementation. For example, NTC1202 references data elements for RS-232 interfaces and Ethernet interfaces. While data elements related to these elements are not mandatory, if a device supports the functionality expressed by them, then they should be supported. This test case specification only deals with the objects that are defined in the MIB. All MIB defined objects are checked for instantiation and the range of values supported.

1.8.1 Requirement Specifications

The requirements specifications are defined in NTCIP 1202 and are summarized in NTCIP 1202 v02.18 - Annex A. The following appears at the beginning of Annex A:

> Conformance Groups are defined as either mandatory or optional. If a Conformance Group is mandatory, all of the objects with STATUS "mandatory" that are part of the Conformance Group shall be present for a device to claim conformance to the Conformance Group. If a Conformance Group is optional, all of the objects that are part of the Conformance Group with the STATUS "mandatory" shall be present if the device supports the Conformance Group. Objects with the STATUS "optional" may be supported.
>
> When a table is included in a Conformance Group, all objects contained in the table are included by reference. This is because a table is defined as a SEQUENCE OF {SEQUENCE}. Thus, all objects listed in the sequence are defined as an integral part of the table. Tables are defined as either mandatory or optional. If a table is mandatory, all of the objects with STATUS "mandatory" shall be present. If a table is optional, all of the objects with the STATUS "mandatory" shall be present if the device supports the table. Objects in the table with the STATUS "optional" may be supported.

The following statements appear in Clause A.2:

> Additional objects or groups may be supported without being non-compliant with ASC objects or NTCIP. Minimum and maximum ranges of objects that differ from the values of the object's SYNTAX field may be enforced by an application running on a device.
>
> A device which enforces range limits within the bounds specified by the values of the object's SYNTAX field shall not be categorized as being non-compliant with ASC objects or NTCIP.
>
> A device which supports a subset of objects with enumerated values shall not be categorized as being non-compliant with ASC objects or NTCIP.

The table in Clause A.2 indicates that the following conformance groups defined within NTCIP 1202 shall be supported:

- Phase Conformance Group
- Detector Conformance Group

The table in Clause A.2 indicates that the following conformance groups defined within other standards shall be supported:

- Configuration Conformance Group
- Database Management Conformance Group

- SNMP Group
- Systems Group
- Security Group

Clause A.2 also indicates that the following conformance groups defined within NTCIP 1202 may be optionally supported:
- Volume Occupancy Report Conformance Group
- Unit Conformance Group
- Special Function Conformance Group
- Coordination Conformance Group
- Time Base Conformance Group
- Preempt Conformance Group
- Ring Conformance Group
- Channel Conformance Group
- Overlap Conformance Group
- TS 2 Port 1 Conformance Group
- Block Object Conformance Group

Clause A.2 also indicates that the following conformance groups defined within other standards may be optionally supported:
- Report Conformance Group
- PMPP Group
- STMP Group
- Logical Name Group
- Trap Management Group
- RS232 Group
- HDLC Group
- Interfaces Group
- IP Group
- ICMP Group
- TCP Group
- UDP Group
- Ethernet Group

The purpose of this test case is to determine what data elements (objects definitions) are supported (instantiated) in an implementation. While NTCIP 1202 is, technically, an information profile that defines the data elements related to an Actuated Signal Controller, it also enumerates other data elements that would be instantiated in a fully conformant implementation. For example, NTC1202 references data elements for RS-232 and Ethernet Interfaces. NTCIP 1202 does not define these but does reference them. While the data elements for RS-232 and Ethernet are not mandatory, if a device supports the functionality expressed by them, then they should be supported. This test case specification not only deals with the objects that are defined in the NTCIP 1202 MIB but also those referenced in other standards.

1.8.2 Design Specifications

There are two perspectives to this test case. What objects does a DUT support and what objects defined in NTCIP 1202 are supported by a DUT.

1. In the context of SNMP, a "MIB Walk" can be used to discover what objects a DUT supports. This set of objects would include not only NTCIP 1202 defined data elements but also "any" data element, be it either NTCIP defined or not. A "Get Next" of the root Object Identifier that defines NTCIP objects can determine what objects are supported. Comparing this list of supported objects to those defined in the standard verifies whether an object is instantiated or not.

2. Given the Object Identifiers of the data elements defined in the standard and the values defined in a DUT to indicate the number of "rows" in certain tables, a list of fully indexed Object Identifiers can be created. A "Get" of each object identifier in the list can verify whether each object is instantiated.

A set of test procedures are to be developed using both of these methods. The test procedures for using the "Get Next" approach shall be referred to as the MIB Walk test case. The procedures using the "Get" approach shall be referred to as the "MIB Check" test case. Since the test procedures are equivalent, either one may be used to verify object instantiation.

1.8.3 User Guide

1.8.4 Operators Guide

1.8.5 Installation Guide

1.9 INPUT SPECIFICATIONS

The requirements as stated in NTCIP 1202 imply that the data elements within appropriate groups shall be instantiated. However, some of the groups and individual data elements are not mandatory.

In NTCIP 1202 v02.18 - Annex A, the table in Clause A.2 indicates whether a conformance group is mandatory or optional. The tables in Clauses A.3 through A.35 enumerate the objects in each conformance group and indicate which specific objects are mandatory or optional.

Note that the list does not add the ".0" in the case of leaf objects or the ".1 .. .X" in the case of columnar objects. Annex B of this document contains a list of objects that contain max values. These objects serve to define the number of rows in various tables. The value of X in the ".1 .. .X" extensions shall match the appropriate max value.

In the case of the MIB Walk test case, there are no input specifications per se. Get Next operations are performed until a noSuchName error is returned.

In the case of the MIB check test case, the input specification is a MIB compiler "tree output" containing all the potential objects that may be supported by the DUT.

1.10 OUTPUT SPECIFICATIONS

The output specification of object instantiation is a "GetResponse" with an Error-Status of noError when a GetRequest of OID is sent to the device under test. A GetResponse with an Error-Status of noSuchName indicates that the object is not supported and should be duly noted.

1.11 ENVIRONMENTAL NEEDS

1.11.1 Hardware

The following is the hardware needed to conduct this test case:

- Actuated Traffic Signal Controller that supports NTCIP 1202 Object Definitions,
- Test software that is capable of generating SNMP GetRequests and SetRequests with the OIDs and values as delineated above, and .
- Test software that is capable of parsing and displaying the values contained in an SNMP GetResponse.

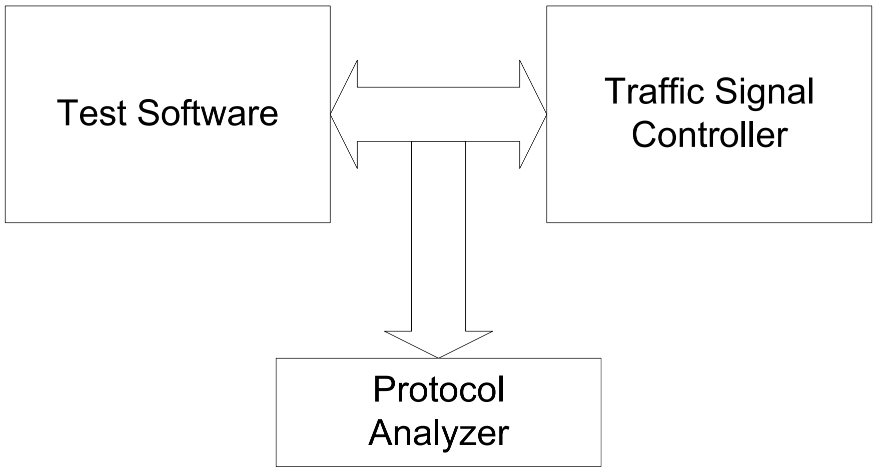
Note - The term "Test software," as used above, is generic. The NTCIP Exerciser, SimpleSoft's SimpleTester™, or another software package that has the ability to generate the interpret SNMP Message may be used.

1.11.2 Software

The communications software in the DUT must be compliant with the communications used in the Test Software.

1.11.3 Other

The following is the configuration of the hardware components:



The use of a protocol analyzer is optional.

1.12 SPECIAL PROCEDURAL REQUIREMENTS

There are no special procedural requirements related to this test case.

1.13 INTERCASE DEPENDENCIES

There are no intercase dependencies related to this test case. The only requirement for reading data is use of the appropriate Community Name. To ensure access to all objects, the community name as defined in communityNameAdmin should be used.

Supported Values Test Case

2.0 TEST ITEMS

This test case applies to devices that implement the data elements defined within NTCIP 1202-ASC. While NTCIP 1202-ASC is, technically, an information profile that specifies all of the data elements to be implemented in an ASC device, this test case specification only deals with the objects that are defined in the MIB. All MIB defined objects are checked for instantiation and the range of values supported.

2.0.1 Requirement Specifications

The requirements specifications are defined in NTCIP 1202 and are summarized in NTCIP 1202 v02.19-Annex A. The following appears at the beginning of Annex A:

> Conformance Groups are defined as either mandatory or optional. If a Conformance Group is mandatory, all of the objects with STATUS "mandatory" that are part of the Conformance Group shall be present for a device to claim conformance to the Conformance Group. If a Conformance Group is optional, all of the objects that are part of the Conformance Group with the STATUS "mandatory" shall be present if the device supports the Conformance Group. Objects with the STATUS "optional" may be supported.
>
> When a table is included in a Conformance Group, all objects contained in the table are included by reference. This is because a table is defined as a SEQUENCE OF {SEQUENCE}. Thus, all objects listed in the sequence are defined as an integral part of the table. Tables are defined as either mandatory or optional. If a table is mandatory, all of the objects with STATUS "mandatory" shall be present. If a table is optional, all of the objects with the STATUS "mandatory" shall be present if the device supports the table. Objects in the table with the STATUS "optional" may be supported.

The following statements appear in Clause A.2:

> Additional objects or groups may be supported without being non-compliant with ASC objects or NTCIP. Minimum and maximum ranges of objects that differ from the values of the object's SYNTAX field may be enforced by an application running on a device.
>
> A device which enforces range limits within the bounds specified by the values of the object's SYNTAX field shall not be categorized as being non-compliant with ASC objects or NTCIP.
>
> A device which supports a subset of objects with enumerated values shall not be categorized as being non-compliant with NTCIP 1202-ASC objects or NTCIP.

The table indicates that the individual data elements in the following conformance groups defined within NTCIP 1202 shall be supported:

- Phase Conformance Group
- Detector Conformance Group

The table also indicates that the individual data elements in the following conformance groups defined within NTCIP 1202 may be optional supported:

- Volume Occupancy Report Conformance Group
- Unit Conformance Group
- Special Function Conformance Group
- Coordination Conformance Group
- Time Base Conformance Group
- Preempt Conformance Group
- Ring Conformance Group

- Channel Conformance Group
- Overlap Conformance Group
- TS 2 Port 1 Conformance Group
- Block Object Conformance Group

It is assumed that test cases and procedures for the other Conformance Groups (and their data elements) are covered in other documents.

2.0.2 Design Specifications

2.0.3 User Guide

2.0.4 Operators Guide

2.0.5 Installation Guide

2.1 INPUT SPECIFICATIONS

The requirements as stated in NTCIP 1202 imply that the data elements within the appropriate group shall be instantiated and that the range or a subrange shall be supported.

A manufacturer's completed PICS should be included in the Test Item Transmittal Report. It should list the upper and lower bounds of an object's supported value range.

Annex B of this document contains some clarification on how to interpret what may be listed in a manufacturer's PICS.

2.2 OUTPUT SPECIFICATIONS

The output specification of the checks for supported values is a "GetResponse" with an Error-Status of noError when a "SetRequest" of the OID is sent to the device under test. Any other Error-Status indicates the value.

2.3 ENVIRONMENTAL NEEDS

2.3.1 Hardware

The following is the hardware needed to conduct this test case:

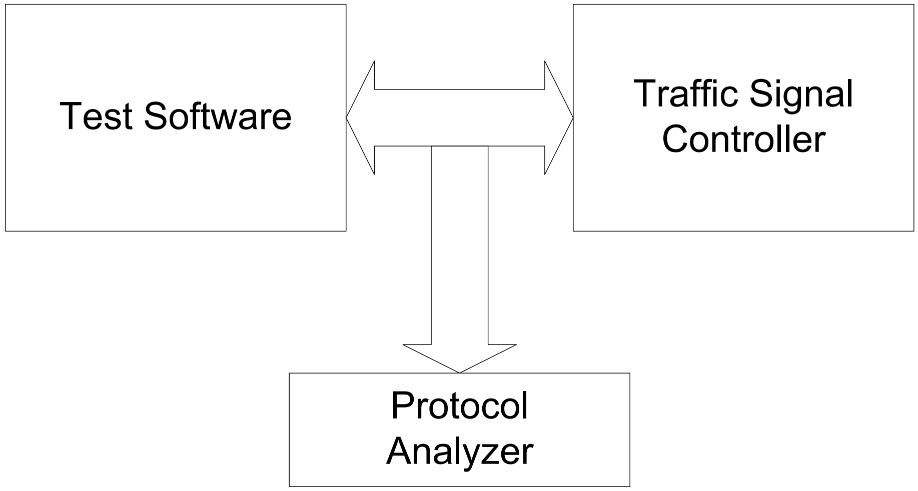
- Actuated Traffic Signal Controller that supports NTCIP 1202 Object Definitions.
- Test software that is capable of generating SNMP GetRequests and SetRequests with the OIDs and values as delineated above.
- Test software that is capable of parsing and displaying the values contained in an SNMP GetResponse.

2.3.2 Software

The communications software in the DUT must be compliant with that used in the Exerciser.

2.3.3 Other

The following is the configuration of the hardware components:



The use of a protocol analyzer is optional.

2.4 SPECIAL PROCEDURAL REQUIREMENTS

There are no special procedural requirements related to this test case.

2.5 INTERCASE DEPENDENCIES

The intercase dependency related to this test case specification is support of the Database Management Conformance Group whose data elements are defined in NTCIP 1201. Testing of "P2" Object Types requires support of dbCreateTransaction.

SNMP Protocol Test Case

3.0 TEST ITEMS

The overall purpose of this test case is to verify conformance to the SNMP Protocol. The SNMP items and features to be exercised by this test case include:

- getNextRequest
- getRequest
- setRequest
- getResponse
- Multiple Variable Binding in a setRequest
- Multiple Variable Binding in a getRequest
- Errors
    - badValue
    - readOnly
    - noSuchName
    - badValue in multiple variable binding
- Invalid communityName
- Data Elements in SNMP Conformance Group

3.1 INPUT SPECIFICATIONS

The inputs specifications for this test case are defined in the General, Get and Set Commands, SNMP Errors, Encoding Rules, and SNMP Configuration Group Sessions of the VDOT- SNMP Test Procedures Report - Rev 1.doc.

3.2 OUTPUT SPECIFICATIONS

The output specifications for this test case are defined in the General, Get and Set Commands, SNMP Errors, Encoding Rules, and SNMP Configuration Group Sessions of the VDOT- SNMP Test Procedures Report - Rev 1.doc.

3.3 ENVIRONMENTAL NEEDS

3.3.1 Hardware

The following is the hardware needed to conduct this test case:

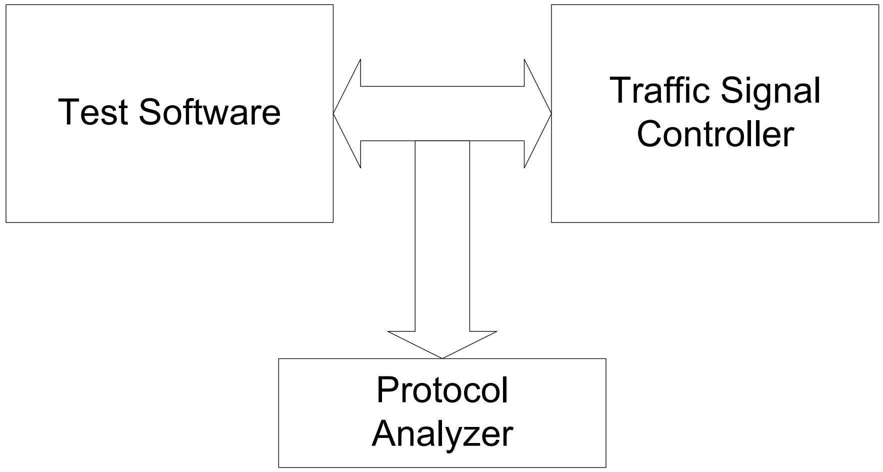
- Actuated Traffic Signal Controller that supports the SNMP portions of NTCIP 2301 Simple Transportation Management Framework Application Profile.
- Test software that is capable of generating SNMP GetRequests and SetRequests with the OIDs and values as delineated above.
- Test software that is capable of parsing and displaying the values contained in an SNMP GetResponse.

3.3.2 Software

The communications software in the DUT must be compliant with that used in the Exerciser.

3.3.3 Other

The following is the configuration of the hardware components:



The use of a protocol analyzer is optional.

3.4 SPECIAL PROCEDURAL REQUIREMENTS

There are no specific special procedural requirements associated with this test case.

3.5 INTERCASE DEPENDENCIES

There are no specific intercase dependencies associated with this test case.

Null Protocol Test Case

4.0 TEST ITEMS

The items and features to be exercised by this test case include:

- Handling of Initial Protocol Identifier

4.1 INPUT SPECIFICATIONS

The inputs specifications for this test case are defined in HDLC Errors Session of the VDOT- SNMP Test Procedures Report - Rev 1.doc.

4.2 OUTPUT SPECIFICATIONS

The output specifications for this test case are defined in HDLC Errors Session of the VDOT- SNMP Test Procedures Report - Rev 1.doc.

4.3 ENVIRONMENTAL NEEDS

4.3.1 Hardware

The following is the hardware needed to conduct this test case:

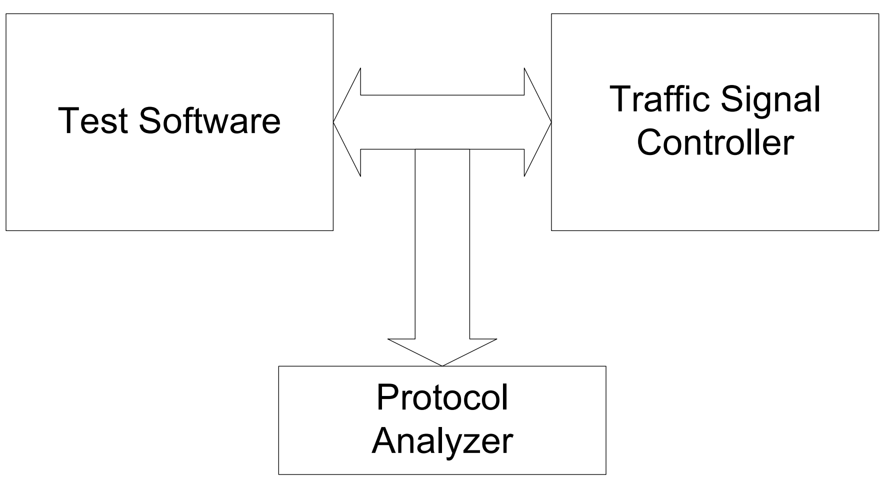
- Actuated Traffic Signal Controller that supports NTCIP 1202 Object Definitions.
- Test software that is capable of generating SNMP GetRequests and SetRequests with the OIDs and values as delineated above.
- Test software that is capable of parsing and displaying the values contained in an SNMP GetResponse.

4.3.2 Software

The communications software in the DUT must be compliant with that used in the Exerciser.

4.3.3 Other

The following is the configuration of the hardware components:



The use of a protocol analyzer is optional.

4.4 SPECIAL PROCEDURAL REQUIREMENTS

There are no specific special procedural requirements associated with this test case.

4.5 INTERCASE DEPENDENCIES

There are no specific intercase dependencies associated with this test case.

PMPP Using RS-232 Test Case

5.0 TEST ITEMS

The items and features to be exercised by this test case include:

- address field
    - one byte form
    - two byte form
- broadcast
- group address
    - one byte form
    - two byte form
- control field
- HDLC errors
    - unknown IPI
    - Invalid CRC value
    - Invalid data stream
- data elements in RS-232 Conformance Group
- data elements in LapB Conformance Group

5.1 INPUT SPECIFICATIONS

The input specifications for this test case are defined in VDOT- Class B Test Procedures - Rev 1.doc.

5.2 OUTPUT SPECIFICATIONS

The output specifications for this test case are defined in VDOT- Class B Test Procedures - Rev 1.doc.

5.3 ENVIRONMENTAL NEEDS

5.3.1 Hardware

The following is the hardware needed to conduct this test case:

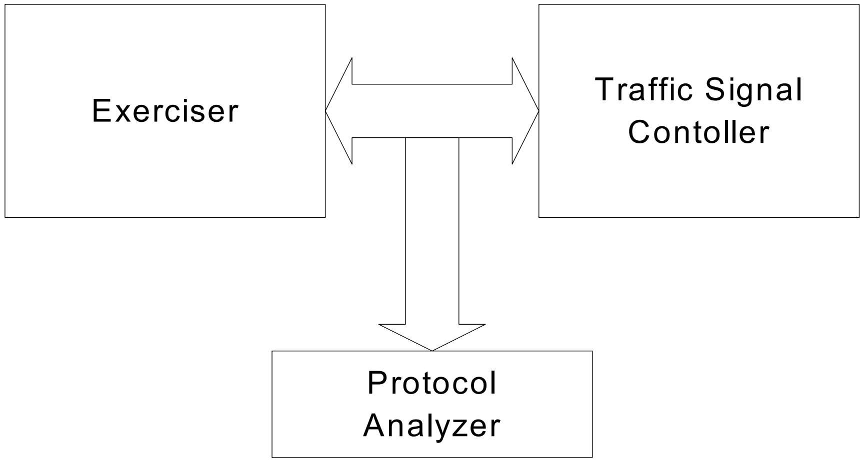
- Actuated Traffic Signal Controller that supports NTCIP 1202 Object Definitions.
- Test software that is capable of generating SNMP GetRequests and SetRequests with the OIDs and values as delineated above.
- Test software that is capable of parsing and displaying the values contained in an SNMP GetResponse.

5.3.2 Software

The communications software in the DUT must be compliant with that used in the Exerciser.

5.3.3 Other

The following is the configuration of the hardware components:



The use of a protocol analyzer is optional.

5.4 SPECIAL PROCEDURAL REQUIREMENTS

There are no specific special procedural requirements associated with this test case.

5.5 INTERCASE DEPENDENCIES

There are no specific intercase dependencies associated with this test case.

PMPP Using FSK Modem Test Case

The tests conducted with respect to the Point-to-Multipoint Protocol using an FSK Modem Interface shall consist of conducting the System Operational Scenarios with a 1200 Bps FSK Modem (see Subclause 1.4.7). This test is contingent upon DUT support of the NTCIP 2102 (PMPP using FSK Modem).

6.0 TEST ITEMS

The items and features to be exercised by this test case include:

- FSK Modem

6.0.1 Requirements specifications

Execution of this test case requires the DUT to support an FSK Modem interface.

6.1 INPUT SPECIFICATIONS

6.2 OUTPUT SPECIFICATIONS

6.3 ENVIRONMENTAL NEEDS

6.3.1 Hardware

The following is the hardware needed to conduct this test case:

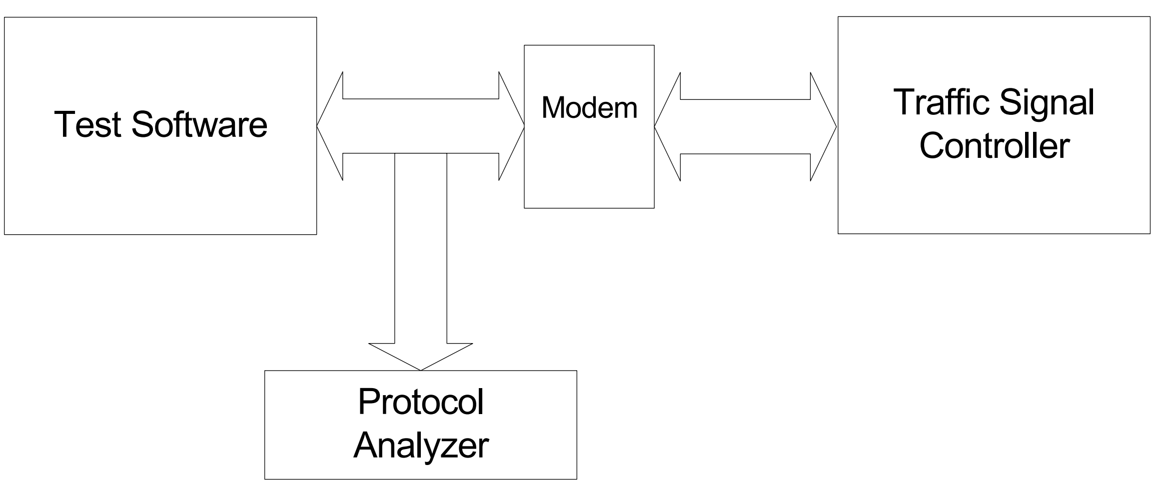
- Actuated Traffic Signal Controller that supports NTCIP 1202 Object Definitions and an FSK Modem Interface.
- An external FSK Modem to be connected to the test software computer.
- Test software that is capable of generating SNMP GetRequests and SetRequests with the OIDs and values as delineated above.
- Test software that is capable of parsing and displaying the values contained in an SNMP GetResponse.

6.3.2 Software

The communications software in the DUT must be compliant with that used in the Exerciser.

6.3.3 Other

The following is the configuration of the hardware components:



The use of a protocol analyzer is optional.

6.4 SPECIAL PROCEDURAL REQUIREMENTS

There are no specific special procedural requirements associated with this test case.

6.5 INTERCASE DEPENDENCIES

There are no specific intercase dependencies associated with this test case.

System Operational Scenarios Test Cases

7.0 TEST ITEMS

The items and features to be exercised by this test case include:

- Setting and verifying the time and date
- Setting up several Dynamic Objects
- Retrieving data typically used to display an intersection map
- Retrieving the status of an eight intersection system
- Two channel operation

7.0.1 Requirements specifications

A. For setting and verifying the time and date, the following NTCIP 1202 objects will be checked for proper operation:
- globalTime
- controller-localTime
- globalDaylightSavings
- controller-standardTimeZone

B. For setting up several Dynamic messages, two dynamic objects will be defined that correspond to the objects used in the following:
- intersection map
- eight intersection system

C. To simulate an eight-phase intersection map, the following NTCIP 1202 objects will be retrieved:
- channelStatusGroupGreens (Group 1)
- channelStatusGroupYellows (Group 1)
- vehicleDetectorStatusGroupActive (Group 1)
- phaseStatusGroupPedCall (Group 1)
- cordPatternStatus
- unitControlStatus
- shortAlarmStatus
- ringStatus (sequenceRingNumber 1&2)

If the DUT supports more than 8 phases, the following objects will be retrieved:
- channelStatusGroupGreens (Groups 1&2)
- channelStatusGroupYellows (Groups 1&2)
- vehicleDetectorStatusGroupActive (Groups 1&2)
- phaseStatusGroupPedCalls (Group 1)
- cordPatternStatus
- unitControlStatus
- shortAlarmStatus
- ringStatus (sequenceRingNumber 1&2)

D. For the status of an eight-intersection system, the following NTCIP 1202 objects will be retrieved from each intersection:
- channelStatusGroupGreens (Group 1)
- vehicleDetectorStatusGroupActive (Group 1)
- shortAlarmStatus

E. Two-channel operation is a test of a management application and therefore will not be included in this test case.

7.0.2 Design Specifications

7.0.3 User Guide

7.0.4 Operators Guide

7.0.5 Installation Guide

7.1 INPUT SPECIFICATIONS

7.2 OUTPUT SPECIFICATIONS

7.3 ENVIRONMENTAL NEEDS

7.3.1 Hardware

The following is the hardware needed to conduct this test case:

- Actuated Traffic Signal Controller that supports NTCIP 1202 Object Definitions.
- Management software that is capable of generating SNMP GetRequests and SetRequests with the OIDs and values as delineated above.
- Management software that is capable of parsing and displaying the values contained in an SNMP GetResponse.

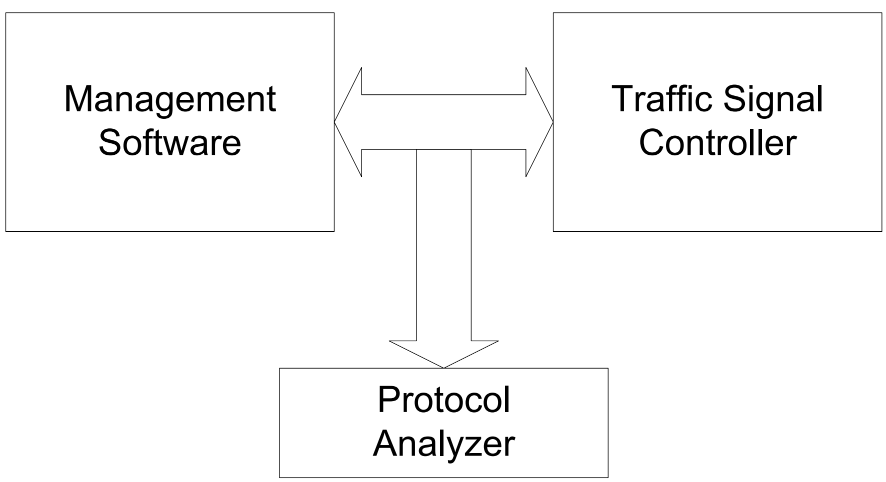
Note - The term "Management software," as used above, is generic. It refers to any PC Software package that has the ability to generate and interpret SNMP messages in the manner described above.

7.3.2 Software

The communications software in the DUT must be compliant with that used in the Management Software.

7.3.3 Other

The following is the configuration of the hardware components:



The use of a protocol analyzer is optional.

7.4 SPECIAL PROCEDURAL REQUIREMENTS

There are no specific special procedural requirements associated with this test case.

7.5 INTERCASE DEPENDENCIES

There are no specific intercase dependencies associated with this test case.

Optional System Operational Scenarios Test Cases - Retrieving Log Data

8.0 TEST CASE SPECIFICATION IDENTIFICATION

8.1 TEST ITEMS

8.1.1 Requirements Specifications

A. For setting and verifying the time and date, the following NTCIP 1202 objects will be checked for proper operation:

- maxEventClasses
- eventClassTable
- eventClassEntry
  - eventClassNumber
  - eventClassLimit
  - eventClassClearTime
  - eventClassDescription
  - eventClassNumRowsInLog
  - eventClassNumEvents
- maxEventLogConfigs
- eventLogConfigTable
- eventLogConfigEntry
  - eventConfigID
  - eventConfigClass
  - eventConfigMode
  - eventConfigCompareValue
  - eventConfigCompareValue2
  - eventConfigCompareOID
  - eventConfigLogOID
  - eventConfigAction
  - eventConfigStatus
- maxEventLogSize
- eventLogTable
- eventLogEntry
  - eventLogClass
  - eventLogNumber
  - eventLogID
  - eventLogTime
  - eventLogValue
- numEvents

8.1.2 Design Specifications

A. The following are the pre-conditions:

- maxEventClasses shall support a minimum value of 6.
- The controller is configured a standard 8-phase quad.
- Controller is cycling through all 8 phases and cycle time should be 100 seconds.
- All eventConfigAction.X are set = disabled(2).
- 5 < phaseMinimumGreen.1 > 15.

B. Configure the time management entries as follows:

globalTime.0	globalDaylightSaving.0	controller-standardTimeZone.0
local time + 21600	enableUSDST	- 21600 (CST = 6 hours)

Note: This sets controller-localTime to current TOD.

C. Configure the entries in the eventClassTable as follows:

eventClassNumber	eventClassLimit	eventClassClearTime	eventClassDescription
1	2	controller-localTime	"Class 1, Limit 2, onChange"
2	3	controller-localTime	"Class 2, Limit 3, greaterThan"
3	4	controller-localTime	"Class 3, Limit 4, smallerThan"
4	5	controller-localTime	"Class 4, Limit 5, hystereis"
5	6	controller-localTime	"Class 5, Limit 6, periodic"
6	7	controller-localTime	"Class 6, Limit 7, andWith"

Note: Setting eventClassClearTime = controller-localTime clears any previous events

D. Configure the entries in eventLogConfigTable as follows:

ID	Class	Mode	Compare Value	Compare Value2	CompareOID	Log OID	Action
1	1	onChange(2)	n/a	n/a	phaseMinimumGreen.1	phaseMinimum Green.1	disabled (2)
2	2	greaterThan(3)	0x44	10	phaseStatusGroupGreens.1	phaseStatus GroupGreens.1	disabled (2)
3	3	smallerThan (4)	0x88	10	phaseControlGroupHold.1	phaseControl GroupHold.1	disabled (2)
4	4	hysteresis (5)	6	7	phaseMinimumGreen.1	phaseMinimum Green.1	disabled (2)
5	5	Periodic (6)	10	n/a	globalTime.0	coordCycle Status.0	disabled (2)
6	6	andedWith (7)	0x11	n/a	phaseStatusGroupPhase Ons.1	phaseStatus GroupPhase Ons.1	disabled (2)

Note: After configuration and having the controller complete 2 complete cycles, is a check of the logs should indicate that they are cleared.

E. For Class 1 Events, configure the entries in eventLogConfigTable as follows:

ID	Class	Mode	Compare Value	Compare Value2	CompareOID	Log OID	Action
1	1	onChange(2)	n/a	n/a	phaseMinimumGreen.1	phaseMinimum Green.1	log (2)

Then perform the following:

Set phaseMinimumGreen.1= phaseMinimumGreen.1 + 1
Wait 15 seconds
Set phaseMinimumGreen.1= phaseMinimumGreen.1
Verify eventClassNumEvents.ClassNumber = 2
Verify numEvents.0 = 2
Verify that:

eventLogClass	eventLogNumber	eventLogID	eventLogTime	eventLogValue
1	1	1	~current local Time - 15	phaseMinimumGreen.1 + 1
1	2	1	~current local Time	phaseMinimumGreen.1

Configure the entries in eventLogConfigTable as follows:

ID	Class	Mode	Compare Value	Compare Value2	CompareOID	Log OID	Action
1	1	onChange(2)	n/a	n/a	phaseMinimum Green.1	phaseMinimum Green.1	disabled(2)

Configure the entries in the eventClassTable as follows:

eventClassNumber	eventClassLimit	eventClassClearTime	eventClassDescription
1	2	current-localTime	"Class 1, Limit 2, onChange"

Verify eventClassNumEvents.1 = 0

F. For Class 2 Events, configure the entries in eventLogConfigTable as follows:

ID	Class	Mode	Compare Value	Compare Value2	CompareOID	Log OID	Action
2	2	greaterThan(3)	0x44	10	phaseStatusGroup Greens.1	phaseStatus GroupGreens.1	log (3)

Then perform the following:
    Wait 330 Seconds
    Verify eventClassNumEvents.2 = 3
    Verify that:

eventLogClass	eventLogNumber	eventLogID	eventLogTime	eventLogValue
2	1	2		0x88
2	2	2	> eventLogTime.1	0x88
2	3	2	> eventLogTime.2	0x88

Configure the entries in eventLogConfigTable as follows:

ID	Class	Mode	Compare Value	Compare Value2	CompareOID	Log OID	Action
2	2	greaterThan(3)	0x44	2	phaseStatusGroup Greens.1	phaseStatusGroup Greens.1	Disabled (2)

Configure the entries in the eventClassTable as follows:

eventClassNumber	eventClassLimit	eventClassClearTime	eventClassDescription
2	3	current-localTime	"Class 2, Limit 3, greaterThan"

Verify eventClassNumEvents.2 = 0

G. For Class 3 Events, configure the entries in eventLogConfigTable as follows:

ID	Class	Mode	Compare Value	Compare Value2	CompareOID	Log OID	Action
3	3	smallerThan (4)	0x01	10	phaseControlGroup Hold.1	phaseControlGroup Hold.1	log (3)

Then perform the following:
    Wait 15 seconds

Set phaseControlGroupHold.1 = 0x01
Wait 15 seconds
Set phaseControlGroupHold.1 = 0x00
Wait 15 seconds
Verify eventClassNumEvents.3 = 3
Verify that:

eventLogClass	eventLogNumber	eventLogID	eventLogTime	eventLogValue
3	1	3	~current local Time -35	0x00
3	2	3	~current local Time -20	0x01
3	3	3	~current local Time – 5	0x00

Configure the entries in eventLogConfigTable as follows:

ID	Class	Mode	Compare Value	Compare Value2	CompareOID	Log OID	Action
3	3	smallerThan (4)	0x88	10	phaseControlGroup Hold.1	phaseControlGroup Hold.1	disabled(2)

Configure the entries in the eventClassTable as follows:

eventClassNumber	eventClassLimit	eventClassClearTime	eventClassDescription
3	4	current localTime	"Class with limit of 4."

Verify eventClassNumEvents.3 = 0

H. For Class 4 Events, configure the entries in eventLogConfigTable as follows:

ID	Class	Mode	Compare Value	Compare Value2	CompareOID	Log OID	Action
4	4	hysteresis (5)	5	15	phaseMinimum Green.1	phaseMinimum Green.1	log (3)

Then perform the following:
Set phaseMinimumGreen.1= 0x04
Wait 15 Seconds
Set phaseMinimumGreen.1= 0x10
Wait 15 Seconds
Set phaseMinimumGreen.1= original value
Wait 15 seconds
Verify eventClassNumEvents.3 = 2
Verify that:

eventLogClass	eventLogNumber	eventLogID	eventLogTime	eventLogValue
4	1	4	~current local Time - 20	0x04
4	2	4	~current local Time - 5	0x10

Configure the entries in eventLogConfigTable as follows:

ID	Class	Mode	Compare Value	Compare Value2	CompareOID	Log OID	Action
4	4	hysteresis (5)	6	7	phaseMinimum Green.1	phaseMinimum Green.1	disabled(2)

Configure the entries in the eventClassTable as follows:

eventClassNumber	eventClassLimit	eventClassClearTime	eventClassDescription
4	4	current localTime	"Class with limit of 5."

Verify eventClassNumEvents.4 = 0

I. For Class 5 Events, configure the entries in eventLogConfigTable as follows:

ID	Class	Mode	Compare Value	Compare Value2	CompareOID	Log OID	Action
5	5	periodic (6)	10	n/a	globaltime.0	coordCycle Status.0	log (3)

Then perform the following:
Wait 125 seconds
Verify eventClassNumEvents.3 = 6
Verify that:

eventLogClass	eventLogNumber	eventLogID	eventLogTime	eventLogValue
5	1	5	~current local Time - 55	~current local Time - 55
5	2	5	~current local Time - 45	~current local Time - 45
5	3	5	~current local Time - 35	~current local Time - 35
5	4	5	~current local Time - 25	~current local Time - 25
5	5	5	~current local Time - 15	~current local Time - 15
5	6	5	~current local Time - 5	~current local Time - 5

Configure the entries in eventLogConfigTable as follows:

ID	Class	Mode	Compare Value	Compare Value2	CompareOID	Log OID	Action
5	5	periodic (6)	10	n/a	globalTime.0	coordCycle Status.0	disabled(2)

Configure the entries in the eventClassTable as follows:

eventClassNumber	eventClassLimit	eventClassClearTime	eventClassDescription
5	4	current localTime	"Class with limit of 5."

Verify eventClassNumEvents.4 = 0

J. For Class 6 Events, configure the entries in eventLogConfigTable as follows:

ID	Class	Mode	Compare Value	Compare Value2	CompareOID	Log OID	Action
6	6	andedWith (7)	0x11	n/a	phaseStatusGroup PhaseOns.1	phaseStatusGroup PhaseOns.1	log (3)

Then perform the following:
Wait 5 minutes
Verify eventClassNumEvents.6 = 6
Verify that:

eventLogClass	eventLogNumber	eventLogID	eventLogTime	eventLogValue
6	1	6	n/a	0x11
6	2	6	n/a	0x11
6	3	6	n/a	0x11
6	4	6	n/a	0x11
6	5	6	n/a	0x11
6	6	6	n/a	0x11

Configure the entries in eventLogConfigTable as follows:

ID	Class	Mode	Compare Value	Compare Value2	CompareOID	Log OID	Action
6	6	andedWith (7)	0x11	n/a	phaseStatusGroup PhaseOns.1	phaseStatusGroup PhaseOns.1	disabled (3)

Configure the entries in the eventClassTable as follows:

eventClassNumber	eventClassLimit	eventClassClearTime	eventClassDescription
6	6	current localTime	"Class with limit of 6."

Verify eventClassNumEvents.4 = 0

Note that the test case of retrieving log data requires support of the NTCIP 1202 v2 Block Object Conformance Groups and the Block Data Type and ID requirements.

Optional System Operational Scenarios Test Cases - Retrieving Log Data

9.0 TEST CASE SPECIFICATION IDENTIFICATION

The test case of setting up a timebase schedule of events requires support of the NTCIP 1202 v2 TimeBlock Object Conformance Groups and the Block Data Type and ID requirements.

The test case of performing the database upload and download requires support of the NTCIP 1202 v2 Block Object Conformance Groups and the Block Data Type and ID requirements.

9.0.1 Design Specifications

9.0.2 User Guide

9.0.3 Operators Guide

9.0.4 Installation Guide

9.1 INPUT SPECIFICATIONS

9.2 OUTPUT SPECIFICATIONS

9.3 ENVIRONMENTAL NEEDS

9.3.1 Hardware

The following is the hardware needed to conduct this test case:

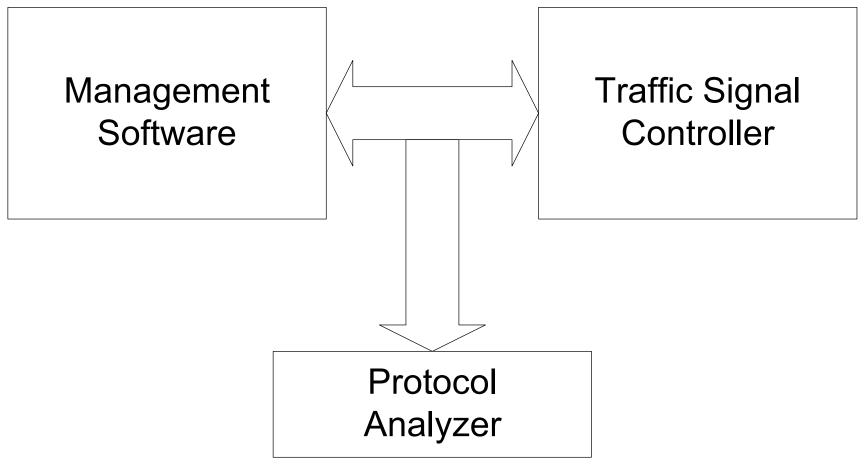
- Actuated Traffic Signal Controller that supports NTCIP 1202 Object Definitions.
- Management Software that is capable of generating SNMP GetRequests and SetRequests with the OIDs and values as delineated above.
- Management Software that is capable of parsing and displaying the values contained in an SNMP GetResponse.

9.3.2 Software

The communications software in the DUT must be compliant with that used in the Exerciser.

9.3.3 Other

The following is the configuration of the hardware components:



The use of a protocol analyzer is optional.

9.4 SPECIAL PROCEDURAL REQUIREMENTS

There are no specific special procedural requirements associated with this test case.

9.5 INTERCASE DEPENDENCIES

There are no specific intercase dependencies associated with this test case.

System Performance Testing Test Case

10.0 TEST CASE SPECIFICATION IDENTIFICATION

10.1 TEST ITEMS

The items and features to be exercised by this test case include:

- Operation of Six-Intersection System using 1200 BPS
- Operation of Six-Intersection System using 9600 BSP

One or more test cases will be defined to maximize the number of messages that can be exchanged when the Subnetwork operates at 1200 BPS. The mix of messages should include single addressed gets and sets and one or more group addressed sets. Given the throughput limitations of 1200 BPS, the STMP Protocol will be used.

One or more test cases will be defined to maximize the number of messages that can be exchanged when the Subnetwork operates at 9600 BPS. The mix of messages should include single addressed gets and sets and one or more group addressed sets. Both the SNMP and STMP Protocols will be used to conduct these tests.

10.1.1 Requirements specifications

One or more test cases will be defined to maximize the number of messages that can be exchanged when the Subnetwork operates at 1200 BPS. The mix of messages should include single addressed gets and sets and one or more group addressed sets. Given the throughput limitations of 1200 BPS, the STMP Protocol will be used.

One or more test cases will be defined to maximize the number of messages that can be exchanged when the Subnetwork operates at 9600 BPS. The mix of messages should include single addressed gets and sets and one or more group addressed sets. Both the SNMP and STMP Protocols will be used to conduct these tests.

10.1.2 Design Specifications

The message used in the Phoenix and Lakewood system will be used:

- Once per second status poll (long and short)
- Command timing plan and special functions
- Upload/download system time and DST flag
- Polling for Detector volume and Occupancy

10.1.3 User Guide

10.1.4 Operators Guide

10.1.5 Installation Guide

10.2 INPUT SPECIFICATIONS

NTCIP 1202 does not define specific requirements in terms of throughput and response time. However, the initial installation of the NTCIP Conformant Traffic Signal Controllers provides some guidance on a systems integrator's expectation. Transcore's "NTCIP Lessons or Making NTCIP Work" (see Lessons Learned Transcore v1.doc) provides specifics about typical messages and timing.

10.3 OUTPUT SPECIFICATIONS

10.4 ENVIRONMENTAL NEEDS

10.4.1 Hardware

The following is the hardware needed to conduct this test case:

- Actuated Traffic Signal Controller that supports NTCIP 1202 Object Definitions.
- Management software that is capable of generating SNMP GetRequests and SetRequests with the OIDs and values as delineated above.
- Management software that is capable of parsing and displaying the values contained in an SNMP GetResponse.
- A suitable modem or RS-232 converters capable of simulating multi-drop.

Depending on the configuration of the DUT, an external modem may be required.

10.4.2 Software

The communications software in the DUT must be compliant with that used in the Exerciser.

10.4.3 Other

The following are two possible configurations for the hardware components of this test case. Figure 1 represents the case where the Traffic Signal Controllers all share a common multi-drop interface.

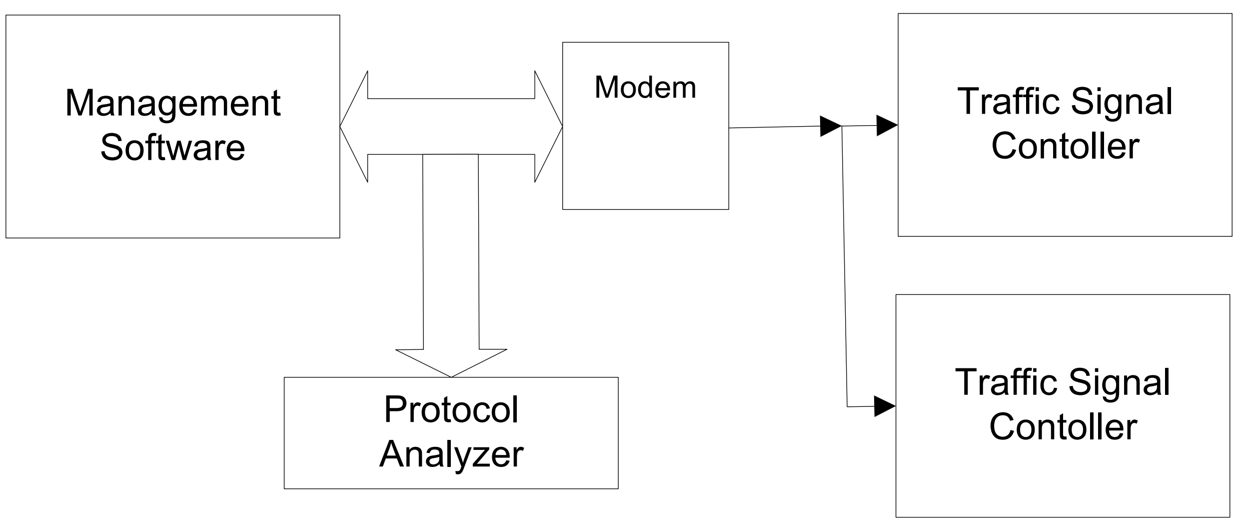


Figure 1

Figure 2 represents an alternate were the RS-232 Interface on the Traffic Signal Controllers is used but converted to multi-drop by some means such as RS-232 to RS-485 Converters.

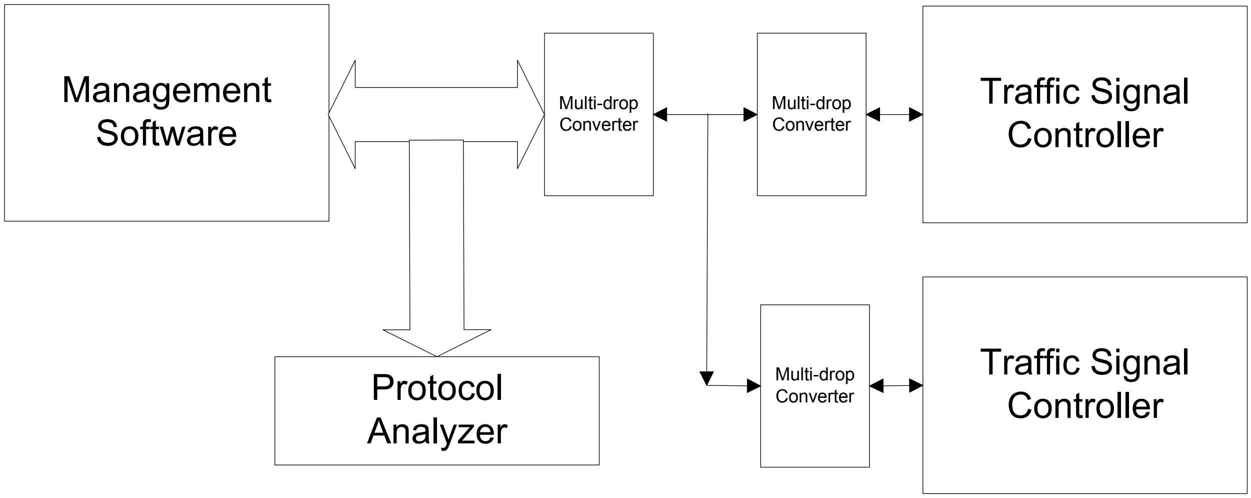


Figure 2

The use of a protocol analyzer is optional.

10.5 SPECIAL PROCEDURAL REQUIREMENTS

There are no specific special procedural requirements associated with this test case.

10.6 INTERCASE DEPENDENCIES

There are no specific intercase dependencies associated with this test case.

ASC Object Identifiers and Names

The following is a list of the Object Identifiers and Object Names that are defined or referenced in NTCIP 1202v02.19.

<Include the latest mib.out file from the SMICng Compiler>

ASC Max Value Object Identifiers

The following is a list of the Object Names defined or referenced in NTCIP 1202 that specify the upper bound or max values for various table indexes. These objects serve to define the number of rows in various tables. For columnar objects, the maximum value of the instance identifiers added to the Object Identifiers for columnar objects shall not exceed the max value.

The following max values are defined in NTCIP 1202:
- maxPhases
- maxPhaseGroups
- maxVehicleDetectors
- maxVehicleDetectorStatusGroups
- maxPedestrianDetectors
- maxAlarmGroups
- maxSpecialFunctionOutputs
- maxPatterns
- maxSplits
- maxTimeBaseScheduleEntries
- maxDayPlans
- maxDayPlanEvents
- maxTimebaseAscActions
- maxPreempts
- maxRings
- maxSequences
- maxRingControlGroups
- maxChannels
- maxChannelStatusGroups
- maxOverlaps
- maxOverlapStatusGroups
- maxPort1Addresses

The following max values are defined in NTCIP 1201 - Global Object Definitions:
- globalMaxModules
- maxTimeBaseScheduleEntries
- maxDayPlans
- maxDayPlanEvents
- maxEventLogConfigs
- maxEventLogSize
- maxEventClasses
- maxGroupAddress

The following max values are defined in NTCIP 1103 - Transportation Management Protocol:
- logicalNameTranslationTable-maxEntries
- communityNamesMax

The following max values are defined in NTCIP 2103 - PPP Subnetwork Profile:
- chapMaxSecrets

The following max values are defined in NTCIP 2202 (RFC 1213) - Internet Transport Profile:
- tcpMaxConn
- ifNumber

The following max values are defined in NTCIP 2101 and 2102 (RFC 1317) - PMPP with RS-232 and FSK Subnetwork Profiles:

- rs232Number

Note: A number of SNMPv1 Interface MIBs do not follow the MIB-II (RFC 1213) convention for identifying table indexes that contain parameters associated with interfaces. Basically, MIB-II assumes that the maximum value of ifIndex is the total number of physical interfaces supported. The value of ifIndex is used to point to one and only one physical interface.

For example, a hypothetical traffic signal controller could have three interfaces that support NTCIP protocols. It might have a RS-232 Interface as its system communications port, an RS-485 Interface as its intra-cabinet communications port, and an Ethernet Interface as its console or laptop communications port. The functionally of the RS-232 and RS-485 interfaces is expressed in RFC 1317 - Definitions of Managed Objects for RS-232-like Hardware Devices. The functionality of the Ethernet Interface is expressed in RFC 1643 - Definitions of Managed Objects for Ethernet-like Interface Types.

In the Ethernet MIB, ifIndex is used as the index in various tables that point to parameters associated with that type of interface. There is no object definition that defines the total number of Ethernet interfaces. In the RS-232 MIB, however, rs232Number defines the total number of ports covered by the MIB and is used as the index into various tables. When RS-232 MIB was re-written to support SNMPv2, however an object still defines the number of ports covered by the MIB but the index into the various tables was changed to ifIndex.

What this means is that even though there could be two RS-232-like interfaces in the hypothetical traffic signal controller, the ifIndex 1 could point to the RS-485 Interface, ifIndex 2 could point to the Ethernet Interface, and ifIndex 3 could point to the RS-232 Interface. There is no NTCIP object definition that maps the ifIndex to the rs232Number. Unless the definition of rs232PortIndex is revised, the value used could be confused with the value of ifIndex.

REFERENCES FOR APPENDIX F

1. IEEE Std 829-1998 – IEEE Standard for Software Test Documentation, Institute of Electrical and Electronics Engineers, New York, New York, 1998.

2. DMS-11170, Fully Actuated, Solid-State Traffic Signal Controller Assembly, Departmental Material Specification, Texas Department of Transportation, August 2006, http://manuals.dot.state.tx.us/dynaweb/colmates/dms/@ebt-link;?target=idmatch(s070019). Accessed July 29, 2006.

3. NTCIP Laboratory Testing for Actuated Signal Controllers, Summary Report for ASSHTO Project 475070. Published by Texas Transportation Institute. http://tti.tamu.edu/documents/TTI-2006-1.pdf. Accessed June 7, 2006.

4. ISO 9000 and ISO 14000 – in brief, ISO 9000, http://www.iso.org/iso/en/iso9000-14000/understand/inbrief.htm. Accessed August 16, 2006.

APPENDIX G:
TRAFFIC SIGNAL CONTROLLER TEST PROCEDURES

INTRODUCTION

This appendix contains a limited set of test procedures for use with NTCIP conformant traffic signal controller field devices. One intent of this research project is to develop a limited set of test procedures for ITS field devices using NTCIP standards to provide cost and effort estimates for creating them for devices in the remaining standardized application areas. Due to the scope and complexity of procedures for traffic signal controllers, the researcher's development focused on procedures that directly relate to TxDOT requirements that would not be part of any outside development. Per agreement with the project monitoring committee, the focus of the research was on testing procedures for the four-phase diamond sequencing and detector operations. Only the details of the prequalification, the four-phase diamond sequencing, and the four-phase detector operations appear after the test case summary. Table G-1 summarizes the test procedure that are a part this research effort.

Table G-2 summarizes some additional test procedures that may be applicable to traffic signal controller testing. However, their development was part of an AASHTO research project (*1*). The Test Design Specification and Test Case Specification in Appendix F includes general background and planning information that relates to development of the these test procedures.

Detector Operations

It is worthy to comment on two issues with respect to the detector operations test procedures. The first issue is that NTCIP does not provide a means to activate detector inputs prior to the operations in a controller that may operate on them. The NTCIP 1202-ASC object phaseControlGroupVehCall places a call for service on a particular phase, but there is no object that acts as a control to activate a detector input (*2*). Detector inputs may undergo memory, switching, delay, and extending operations prior to their routing to a phase. Since the detector operations tests involve phase switching, automating the testing process would be impossible without a means of activating detector inputs through software. A test script interface to the Hardware-in-the-Loop (HITL) software resolves the issue (*3*).

A "Set HITL Detector Input X = On/Off" instruction in the test procedures, sends an operating system inter-processor communications command to the HITL software to activate or deactivate a detector input. The HITL software, in turn, sends the command to the software that controls a TS2 Test Box. The TS2 Test Box uses the Port 1 communication interface of a controller to activate the detector input of the traffic signal controller. There are delays and timing issues to consider when using a HITL interface, but it does provide a solution. The potential for using HITL in testing to provide independent verification of controller operation is worthy of further research. One does not have to rely on visual inspection or retrieval of NTCIP status information to confirm that some action takes place.

The second issue with the detector operations test procedures relates to risk management. A requirement for most of the four-phase diamond detectors is that they operate as both calling and extending detectors. In this operating mode, a detector extends, for example phase 1, when phase 1 is green. The detector also places a call on phase 1 when it is not in phase 1 green. Not in phase 1 means that it could be in phase 2, 3, 4, 5, 6, 7, or 8, for example. Rather than developing a test procedure that samples only some of the other possible states, the test procedures in this appendix look at all of the other intervals. In the case of the four-phase diamond operation, there are 12 distinct states. To ensure that 10 calling detectors operate correctly in the 11 other possible states requires 110 tests. If one considers that a phase can be in one of three possible substates (green, yellow, and red), there could be 330 possible states.

For the purpose of this research project, the researcher chose the more extensive 110-test approach rather that a sampling approach. One test procedure addresses the substates but the rest look at operation during the green only. The number of resulting test steps illustrates the impact of full coverage testing. Of the 183 pages needed to document the 110-test approach, a test procedure that chooses just one of the possible 11 phase states randomly would have reduced the documentation and resulting effort to implement the tests by at least 75%.

A sampling approach runs the risk of not detecting a potential problem in some situations. The probability is that in the long term, it would show up after some number of testing sessions. As it turns out, the extensive testing approach did uncover a problem immediately. Two detectors fail to call the designated phase when in the green of one particular other phase. Both detectors operate per the requirements. The physical geometry of an intersection or signage may preclude the need to call the designated phase in this particular situation. In the testing, however, it does not appear to be correct. A recommendation of the research is to investigate this further.

TEST CASE SUMMARY

Table G-1 provides a summary of the traffic signal controller test procedures and test cases organized by feature or functional area derived from NTCIP 1202-ASC and DMS 11170-TSC (2,4). Two procedures, Global Configuration and Security, have a reference in NTCIP 1202-ASC, but the object definitions appear in NTCIP 1201-GLO (2,5). The test procedures do not address all the functional requirements in DMS 11170-TSC (4).

Table G-1. Traffic Signal Controller Test Case Summary.

Traffic Signal Controller Test Cases		
ID	**Title**	**Description**
Prequalification (PRL)		
TC001	ASC PRL Information	This procedure retrieves minimum project requirements and maximum values, checks for whether the required objects are implemented, and performs a sampling of the supported values.
Four-Phase Diamond Sequencing (Seq)		
TC001	Sequencing	Tests sequencing of all transitions from one state to another using vehicle calls (not detector calls).
Four-Phase Diamond Detector Operations (DetOps)		
TC001	Detector 1 Operations	Verifies the operation of Detector 1 to call Phase 6 under specific conditions and extend intervals 2516B, 2517B, 2518B, 4517B, 4518B, 1517B, and 3518B.
TC002	Detector 2 Operations	Verifies the operation of Detector 2 to call and extend Phase 2. The test for the two-second delay is in TC021.
TC003	Detector 3 Operations	Verifies the operation of Detector 3 to call and extend Phase 3 under specific conditions and to extend interval 3516B. The test for the two-second delay is in TC021.
TC004	Detector 4 Operations	Verifies the operation of Detector 4 to call and extend Phase 4 under specific conditions and extend interval 4516B. The test for the two-second delay is in TC021.
TC005	Detector 5 Operations	Verifies the operation of Detector 5 to call Phase 2 under specific conditions and extend intervals 1625B, 1635, 1645B, 1735B, 1745B, 1835B, and 1845B.

Table G-1. Traffic Signal Controller Test Case Summary (continued).

Traffic Signal Controller Test Cases		
ID	**Title**	**Description**
TC006	Detector 6 Operations	Verifies the operation of Detector 6 to call and extend Phase 6. The test for the two-second delay is in TC021.
TC007	Detector 7 Operations	Verifies the operation of Detector 7 to call and extend Phase 7 under specific conditions and extend interval 1725B. The test for the two-second delay is in TC021.
TC008	Detector 8 Operations	Verifies the operation of Detector 8 to call and extend Phase 8 under specific conditions and extend interval 1825B. The test for the two-second delay is in TC021.
TC009	Detector 9 Operations	Verifies the operation of Detector 9 to call Phase 6 under specific conditions, extend Phase 2 under specific conditions, and extend intervals 2516B, 2517B, 2518B, 3517B, 3518B, 4517B, and 4518B.
TC010	Detector 10 Operations	Verifies the operation of Detector 10 to call Phase 6 under specific conditions, extend Phase 2 under specific conditions, and extend intervals 2516B, 2517B, 2518B, 3517B, 3518B, 4517B, and 4518B.
TC011	Detector 11 Operations	Verifies the operation of Detector 11 to call and extend Phase 2 under specific conditions
TC012	Detector 12 Operations	Verifies the operation of Detector 12 to call and extend Phase 4 under specific conditions and extend interval 4516B (6+12).
TC013	Detector 13 Operations	Verifies the operation of Detector 13 to call Phase 2 under specific conditions, extend Phase 6 under specific conditions, and extend intervals 1625B, 1635B, 1645B, 1735B, 1745B, 1835B, and 1845B.
TC014	Detector 14 Operations	Verifies the operation of Detector 14 to call Phase 2 under specific conditions, extend Phase 6 under specific conditions, and extend intervals 1625B, 1635B, 1645B, 1735B, 1745B, 1835B, and 1845B.
TC015	Detector 15 Operations	Verifies the operation of Detector 15 to call and extend Phase 6 under specific conditions.
TC016	Detector 16 Operations	Verifies the operation of Detector 16 to call and extend Phase 8 under specific conditions and extend interval 1825B.
TC017	Detector 17 Operations	Verifies the operation of Detector 17 to call and extend Phase 3 under specific conditions and extend interval 3516B.

Table G-1. Traffic Signal Controller Test Case Summary (continued).

Traffic Signal Controller Test Cases		
ID	**Title**	**Description**
TC018	Detector 18 Operations	Verifies the operation of Detector 18 to call and extend Phase 7 under specific conditions and extend interval 1725B.
TC019	Detector Operations Setup	This procedure performs general setup of controller parameters to facilitate testing and provide consistent operation.
TC020	Detector Operations Teardown	This procedure restores original controller parameters after executing Detector Operation Setup – TC001.
TC021	Detector Delay	Verifies that, when programmed, Detectors 2, 3, 4, 6, 7, and 8 delay entering a call for the parent phase when the parent phase is red.
Global Configuration (GloCon)		
TC001	Retrieve Module Table	This procedure retrieves the module table, and allows the Tester to verify that the device under test (DUT) reports the proper type of device, manufacturer, model, and version.
TC002	Global Set ID	This procedure ensures that a change to a static database object produces a change in globalSetIDParmeter.
Security		
TC001	Change Administrator Community Name	Verifies that the administrator can change the administrator community name stored in the DUT and properly affects operations.
TC002	Change User Community Name	Verifies that the administrator can change the user community names and their masks stored in the DUT and properly affects operations.

The details of the Global Configuration and Security test cases can be found in Appendix C of this report.

Some additional test procedures that may be applicable to traffic signal controller testing are summarized in Table G-2. However, these test procedures were developed as part of an AASHTO research project (*1*). The details of each procedure and the associated test scripts are available at www.itstestlab.org. The Test Design Specification and Test Case Specification in Appendix F include discussion on the premise and background information on these test procedures.

Table G-2. Additional Traffic Signal Controller Test Case Summary.

Additional Traffic Signal Controller Test Cases		
ID	**Title**	**Description**
dbCreateTransaction (DCT)		
TC0001	dbCreateTransaction	Verifies that the dbCreateTransaction transitions from all states to another properly, object values are buffered when in the transaction state, and a consistency check is performed.
Intersection Map (IM)		
TC0001	Intersection Map	Tests whether controller can produce the phase related parameters of a typical eight-phase, dual ring controller.
Global Configuration (GloCon)		
TC0001	Retrieve Module Table	This procedure retrieves the module table, and allows the Tester to verify that the DUT reports the proper type of device, manufacturer, model, and version.
TC0002	Global Set ID	This procedure ensures that a change to a static database object produces a change in globalSetIDParmeter.
Retrieve Log Data (RLD)		
TC0001	Setup Classes and Configure Events	This procedure checks whether objects are instantiated, sets up the eventClassTable and the eventLogConfigTable.
TC0002	Check Class 1 Events	This procedure checks for proper functioning of the “onChange” eventLogConfig.Mode and whether the resulting log entries are produced.
TC0003	Check Class 2 Events	This procedure checks for proper functioning of the “greaterThan” eventLogConfig.Mode and whether the resulting log entries are produced.
TC0004	Check Class 3 Events	This procedure checks for proper functioning of the “smallerThan” eventLogConfig.Mode and whether the resulting log entries are produced.
TC0005	Check Class 4 Events	This procedure checks for proper functioning of the “hysteresis” eventLogConfig.Mode and whether the resulting log entries are produced.
TC0006	Check Class 5 Events	This procedure checks for proper functioning of the “periodic” eventLogConfig.Mode and whether the resulting log entries are produced.
TC0007	Check Class 6 Events	This procedure checks for proper functioning of the “andedWith” eventLogConfig.Mode and whether the resulting log entries are produced.

Table G-2. Additional Traffic Signal Controller Test Case Summary (continued).

Additional Traffic Signal Controller Test Cases		
ID	**Title**	**Description**
TC0008	Class Events Cleanup	This procedure clears the events in the logs and resets the eventClassDescriptions to null.
System Map (SM)		
TC0001	System Map	The purpose of this test procedure is to demonstrate and verify the retrieval of status information that would typically be viewed in a system map display.
Timebase Schedule of Events (TBE)		
TC0001	Setup[1]	This procedure is used to set up a scheduling configuration.
TC0002	Trigger Scheduled Events	This procedure tests whether the schedule of events occurs.
Database Upload and Download (DUD)		
TC0001	Setup	This procedure verifies that the DUT supports the ascBlock objects, verifies that the administrator can change the administrator community name stored in the DUT, and properly affects operations.
TC0002	Upload ascBlocks	This procedure verifies that a single block of all 21 types can be uploaded.
TC0003	Download ascBlocks	This procedure verifies that a single block of all 21 types can be downloaded.
TC0004	Upload Multiple ascBlocks	This procedure verifies that multiple blocks of all 21 types can be uploaded.
TC0005	Download Multiple ascBlocks	This procedure verifies that multiple blocks of all 21 types can be downloaded.
Time and Date (TAD)		
TC0001	Time and Date[1]	The following procedures check for proper operation in regard to setting and displaying time and date, whether time is expressed as a counter (sign bit is ignored and unit transitions from 0x7F FF FF FF to 0x80 00 00 00). These procedures also check for proper operation when daylight savings is not enabled and when it is. Both transitions (spring ahead and fall back) are checked. Finally, the procedure checks that time zone value has the appropriate effect on controller local time. Since the overflow condition, transitioning from 0xFF FF FF FF to 0x00 00 00 00 occurs in the next century, it is not checked.

[1] Scripts for both globalLocalTimeDifferential and controllStandardTimeZone are available.

TEST CASES

The details of the four-phase diamond sequencing and detector operations test cases follow. The basic format of the test cases comes from the template that appears in NTCIP 8007 – Testing and Conformity Assessment Documentation within NTCIP Standards Publications (6). As presented here the test case format has additional fields for clarity and version control.

ASC PRL Information

Test Case: **TC001**	*Title*: **ASC PRL Information**	
	Description: This procedure retrieves minimum project requirements and maximum values, checks for whether the required objects are implemented, and performs a sampling of the supported values for P and C objects.	
	Constants:	
	Variables:	
	Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
1. (Static Values)	CONFIGURE a list <ASC Max and Static OIDs> that identifies read-only objects that define either maximum values that affect the indexes of tables or static variables that affect limits on other variables	
2.	CONFIGURE COMMUNITY NAME OUT = “administrator“	
3.	FOR ObjectName = each objectName in < ASC Max and Static OIDs>	
4.	GET ObjectName	Pass/Fail
5.	Record the value on the PRL	
6.	NEXT ObjectName	
7. (Object Support)	CONFIGURE a list of objectNames that must be supported	
8.	FOR [objectName = each objectName in Supported objectName List	
9.	FOR all possible [instance values]	
10.	GET [objectName].instance *Note:* This loop performs the equivalent of a MIB Walk but uses GET instead of GET-NEXT.	Pass/Fail
11.	NEXT [instance value]	
12.	NEXT [objectName]	
13. (Range Support)	CONFIGURE a list <ASC Test Values> that identifies instances of objects to test and a value in which to test the object with	
14.	FOR [objectNameInstance] = each objectNameInstance in <CCTV Test Values>	
15.	GET [objectNameInstance]	Pass/Fail
16.	RECORD RESPONSE VALUE in [currentValue]	
17.	FOR [testValue] = each objectNameValue in <ASC Test Values>	
18.	SET [objectNameInstance] = [testValue]	Pass/Fail
19.	Record ObjectName, TestValue, and errorStatus	
20.	NEXT [testValue]	

21.	SET [objectNameInstance] = [currentValue]	Pass/Fail
22.	NEXT ObjectName	
23.	RECORD responses on the associated NTCIP PRL and note any anomalies.	
Test Case Results		
Tested By:	Date Tested	Pass/Fail
Test Case Notes:	<notes>	
Version History:	v1.00 04/05/06 Initial draft – RDR v1.01 07/05/06 Updated notes – RDR v1.02 07/19/06 Implemented script and proofed – JJ	

Four-Phase Diamond Sequencing

Test Case: **TC001**	*Title*: **Sequencing** Description: Tests sequencing of all transitions from one state to another using vehicle calls (not detector calls). Constants: None Variables: [currentMinGrn.Phase] [currentPassage.Phase] [currentMax1.Phase] Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
Setup		
1.	CONFIGURE the controller for: 1. 4-phase diamond operation with Phase 3 and 7 in sequence 2. Connection to TS 2 Tester Box (BIU's enabled) 3. Vehicle and Pedestrian Recalls are all off 4. Default data loaded *Note:* The setup for different manufacturer controllers will be different. Reference should be made to a document containing such information. Eagle Configuration Notes: Unit Data – Startup & Misc – Alt Sequence = 16 Unit Data – Port 1 Data = Enable T&F 1=4, DET 1-4, and Malfunction Unit Phase Data – Initialization & N.A. Response Phase 4 and 7 = Change "Dark" TO "Inactive" Naztec Configuration Notes: Econolite Notes:	
2.	FOR Phase = 1 TO 16	
3.	GET phaseMinimumGreen.Phase, phasePassage.Phase, and phaseMaximum1.Phase	

4.	RECORD RESPONSE VALUE in [currentMinGrn.Phase], [currentPassage.Phase] and [currentMax1.Phase] *Note:* These values will be restored at the end of the test case.	
5.	SET phaseMinimumGreen.Phase = 1, phasePassage.Phase = 10, and phaseMaximum1.Phase = 5	
6.	NEXT Phase	
Sequence from 2+5 with Calls on 2 and 3 = 3+5 and 2+5		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 2 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x12 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 2+5.	
13.	Set HITL Detector Input 2 = Off	
14.	DELAY .2 Seconds	
15.	GET ringStatus.1, ringStatus.2	
16.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
17.	DELAY 1 Second	
18.	GET ringStatus.1, ringStatus.2	
19.	WEND *Note:* Wait for 2+5 Green Rest.	
20.	Set HITL Detector Input 3 = On	
21.	DELAY .2 Seconds	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WHILE phaseStatusGroupPhaseOns.1= 0x12 AND phaseStatusGroupPhaseOns.2= 0x00	
24.	DELAY 1 Second	
25.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
26.	WEND *Note:* Wait until a change from 2+5.	
27.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1= 0x14 AND phaseStatusGroupPhaseOns.2 = 0x00 *Note:* When it does change, it should change to 3+5.	Pass/Fail
28.	Set HITL Detector Input 3 = Off	
29.	DELAY .2 Seconds	
30.	Set HITL Detector Input 2 = On	
31.	DELAY .2 Seconds	
32.	IF phaseStatusGroupPhaseOns.1≠ 0x14 OR	

	phaseStatusGroupPhaseOns.2 ≠ 0x00 THEN GOTO TerminationRestore *Note:* If it does not go to 3+5 then restore original values and then exit.	
33.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
34.	WHILE phaseStatusGroupPhaseOns.1 = 0x14 AND phaseStatusGroupPhaseOns.2 = 0x00	
35.	DELAY 1 Second	
36.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
37.	WEND *Note:* Wait until a change from 3+5.	
38.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1= 0x12 AND phaseStatusGroupPhaseOns.2 = 0x00 *Note:* When it does change, it should change to 2+5.	Pass/Fail
39.	Set HITL Detector Input 2 = Off	
40.	DELAY .2 Seconds	
41.	IF phaseStatusGroupPhaseOns.1≠ 0x12 OR phaseStatusGroupPhaseOns.1≠ 0x00 THEN GOTO TerminationRestore *Note:* If it does not go to 2+5 then restore original values and then exit.	
Sequence from 2+5 with Calls on 2 and 4 = 4+5 and 2+5		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 2 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x12 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 2+5.	
13.	Set HITL Detector Input 2 = Off	
14.	DELAY .2 Seconds	
15.	GET ringStatus.1, ringStatus.2	
16.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
17.	DELAY 1 Second	
18.	GET ringStatus.1, ringStatus.2	
19.	WEND *Note:* Wait for 2+5 Green Rest.	
20.	Set HITL Detector Input 4 = On	

21.	DELAY .2 Seconds	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WHILE phaseStatusGroupPhaseOns.1= 0x12 AND phaseStatusGroupPhaseOns.1= 0x00	
24.	DELAY 1 Second	
25.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
26.	WEND *Note:* Wait until a change from 2+5.	
27.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x18 AND phaseStatusGroupPhaseOns.2 = 0x00 *Note:* When it does change, it should change to 4+5.	Pass/Fail
28.	Set HITL Detector Input 4 = Off	
29.	DELAY .2 Seconds	
30.	Set HITL Detector Input 2 = On	
31.	DELAY .2 Seconds	
32.	IF phaseStatusGroupPhaseOns.1 ≠ 0x18 OR phaseStatusGroupPhaseOns.2 ≠ 0x00 THEN EXIT (Procedure) *Note:* If it does not go to 4+5, then restore original values and exit.	
33.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
34.	WHILE phaseStatusGroupPhaseOns.1 = 0x18 AND phaseStatusGroupPhaseOns.2 = 0x00	
35.	DELAY 1 Second	
36.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
37.	WEND *Note:* Wait until a change from 4+5.	
38.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1≠ 0x12 AND phaseStatusGroupPhaseOns.1≠ 0x00 *Note:* When it does change, it should be to 2+5.	Pass/Fail
39.	Set HITL Detector Input 2 = Off	
40.	DELAY .2 Seconds	
41.	IF phaseStatusGroupPhaseOns.1≠ 0x12 OR phaseStatusGroupPhaseOns.1≠ 0x00 THEN GOTO Termination Restore *Note:* If it does not go to 2+5, then restore original values and then exit.	
Sequence from 2+5 with Calls on 2 and 6 = 5+9, 1+6, 1+13, and 2+5		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 2 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x12 AND	

	phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 2+5.	
13.	Set HITL Detector Input 2 = Off	
14.	DELAY .2 Seconds	
15.	GET ringStatus.1, ringStatus.2	
16.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
17.	DELAY 1 Second	
18.	GET ringStatus.1, ringStatus.2	
19.	WEND *Note:* Wait for 2+5 Green Rest.	
20.	Set HITL Detector Input 6 = On	
21.	DELAY .2 Seconds	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WHILE phaseStatusGroupPhaseOns.1= 0x12 AND phaseStatusGroupPhaseOns.1= 0x00	
24.	DELAY 1 Second	
25.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
26.	WEND *Note:* Wait until a change from 2+5.	
27.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x10 AND phaseStatusGroupPhaseOns.2 = 0x01 *Note:* When it does change, it should change to 5+9.	Pass/Fail
28.	IF phaseStatusGroupPhaseOns.1 ≠ 0x10 OR phaseStatusGroupPhaseOns.2 ≠ 0x01 THEN GOTO TerminationRestore *Note:* If it does not go to 5+9 then restore original values and then exit.	
29.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
30.	WHILE phaseStatusGroupPhaseOns.1= 0x10 AND phaseStatusGroupPhaseOns.1= 0x01	
31.	DELAY 1 Second	
32.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
33.	WEND *Note:* Wait until a change from 5+9.	
34.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x21 AND phaseStatusGroupPhaseOns.2 = 0x00 *Note:* When it does change, it should change to 1+6.	Pass/Fail
35.	Set HITL Detector Input 6 = Off	
36.	DELAY .2 Seconds	
37.	Set HITL Detector Input 2 = On	
38.	DELAY .2 Seconds	
39.	IF phaseStatusGroupPhaseOns.1 ≠ 0x21 OR phaseStatusGroupPhaseOns.2 ≠ 0x00 THEN GOTO TerminationRestore	

	Note: If it does not go to 1+6 then restore original values and then exit.	
40.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
41.	WHILE phaseStatusGroupPhaseOns.1 = 0x21 AND phaseStatusGroupPhaseOns.2 = 0x00	
42.	DELAY 1 Second	
43.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
44.	WEND *Note:* Wait until a change from 1+6.	
45.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1≠ 0x01 AND phaseStatusGroupPhaseOns.1≠ 0x10 *Note:* When it does change, it should be to 1+13.	Pass/Fail
46.	IF phaseStatusGroupPhaseOns.1≠ 0x01 OR phaseStatusGroupPhaseOns.1≠ 0x10 THEN GOTO TerminationRestore *Note:* If it does not go to 1+13 then restore original values and then exit.	
47.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
48.	WHILE phaseStatusGroupPhaseOns.1 = 0x01 AND phaseStatusGroupPhaseOns.2 = 0x10	
49.	DELAY 1 Second	
50.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
51.	WEND *Note:* Wait until a change from 1+13.	
52.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1≠ 0x12 AND phaseStatusGroupPhaseOns.1≠ 0x00 *Note:* When it does change, it should be to 2+5.	Pass/Fail
53.	Set HITL Detector Input 2 = Off	
54.	DELAY .2 Seconds	
55.	IF phaseStatusGroupPhaseOns.1≠ 0x12 OR phaseStatusGroupPhaseOns.1≠ 0x00 THEN GOTO TerminationRestore *Note:* If it does not go to 2+5 then restore original values and then exit.	
Sequence from 2+5 with Calls on 2 and 7 = 5+9, 1+7, 2+15, and 2+5		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 2 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x12 AND	

	phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 2+5.	
13.	Set HITL Detector Input 2 = Off	
14.	DELAY .2 Seconds	
15.	GET ringStatus.1, ringStatus.2	
16.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
17.	DELAY 1 Second	
18.	GET ringStatus.1, ringStatus.2	
19.	WEND *Note:* Wait for 2+5 Green Rest.	
20.	Set HITL Detector Input 7 = On	
21.	DELAY .2 Seconds	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WHILE phaseStatusGroupPhaseOns.1= 0x12 AND phaseStatusGroupPhaseOns.1= 0x00	
24.	DELAY 1 Second	
25.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
26.	WEND *Note:* Wait until a change from 2+5.	
27.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x10 AND phaseStatusGroupPhaseOns.2 = 0x01 *Note:* When it does change, it should change to 5+9.	Pass/Fail
28.	IF phaseStatusGroupPhaseOns.1 ≠ 0x10 OR phaseStatusGroupPhaseOns.2 ≠ 0x01 THEN GOTO TerminationRestore *Note:* If it does not go to 5+9 then restore original values and then exit.	
29.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
30.	WHILE phaseStatusGroupPhaseOns.1= 0x10 AND phaseStatusGroupPhaseOns.1= 0x01	
31.	DELAY 1 Second	
32.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
33.	WEND *Note:* Wait until a change from 5+9.	
34.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x41 AND phaseStatusGroupPhaseOns.2 = 0x00 *Note:* When it does change, it should change to 1+7.	Pass/Fail
35.	Set HITL Detector Input 7 = Off	
36.	DELAY .2 Seconds	
37.	Set HITL Detector Input 2 = On	
38.	DELAY .2 Seconds	
39.	IF phaseStatusGroupPhaseOns.1 ≠ 0x41 OR phaseStatusGroupPhaseOns.2 ≠ 0x00 THEN GOTO TerminationRestore	

	Note: If it does not go to 1+7 then restore original values and then exit.	
40.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
41.	WHILE phaseStatusGroupPhaseOns.1 = 0x41 AND phaseStatusGroupPhaseOns.2 = 0x00	
42.	DELAY 1 Second	
43.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
44.	WEND *Note:* Wait until a change from 1+7.	
45.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1≠ 0x02 AND phaseStatusGroupPhaseOns.1≠ 0x40 *Note:* When it does change, it should be to 2+15.	Pass/Fail
46.	IF phaseStatusGroupPhaseOns.1≠ 0x02 OR phaseStatusGroupPhaseOns.1≠ 0x40 THEN GOTO TerminationRestore *Note:* If it does not go to 2+15 then restore original values and then exit.	
47.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
48.	WHILE phaseStatusGroupPhaseOns.1 = 0x02 AND phaseStatusGroupPhaseOns.2 = 0x40	
49.	DELAY 1 Second	
50.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
51.	WEND *Note:* Wait until a change from 2+15.	
52.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1≠ 0x12 AND phaseStatusGroupPhaseOns.1≠ 0x00 *Note:* When it does change, it should be to 2+5.	Pass/Fail
53.	Set HITL Detector Input 2 = Off	
54.	DELAY .2 Seconds	
55.	IF phaseStatusGroupPhaseOns.1≠ 0x12 OR phaseStatusGroupPhaseOns.1≠ 0x00 THEN GOTO TerminationRestore *Note:* If it does not go to 2+5 then restore original values and then exit.	
Sequence from 2+5 with Calls on 2 and 8 = 5+9, 1+8, 2+16, and 2+5		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 2 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x12 AND	

	phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 2+5.	
13.	Set HITL Detector Input 2 = Off	
14.	DELAY .2 Seconds	
15.	GET ringStatus.1, ringStatus.2	
16.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
17.	DELAY 1 Second	
18.	GET ringStatus.1, ringStatus.2	
19.	WEND *Note:* Wait for 2+5 Green Rest.	
20.	Set HITL Detector Input 8 = On	
21.	DELAY .2 Seconds	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WHILE phaseStatusGroupPhaseOns.1= 0x12 AND phaseStatusGroupPhaseOns.1= 0x00	
24.	DELAY 1 Second	
25.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
26.	WEND *Note:* Wait until a change from 2+5.	
27.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x10 AND phaseStatusGroupPhaseOns.2 = 0x01 *Note:* When it does change, it should change to 5+9.	Pass/Fail
28.	IF phaseStatusGroupPhaseOns.1 ≠ 0x10 OR phaseStatusGroupPhaseOns.2 ≠ 0x01 THEN GOTO TerminationRestore *Note:* If it does not go to 5+9 then restore original values and then exit.	
29.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
30.	WHILE phaseStatusGroupPhaseOns.1 = 0x10 AND phaseStatusGroupPhaseOns.1 = 0x01	
31.	DELAY 1 Second	
32.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
33.	WEND *Note:* Wait until a change from 5+9.	
34.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x81 AND phaseStatusGroupPhaseOns.2 = 0x00 *Note:* When it does change, it should change to 1+8.	Pass/Fail
35.	Set HITL Detector Input 8 = Off	
36.	DELAY .2 Seconds	
37.	Set HITL Detector Input 2 = On	
38.	DELAY .2 Seconds	
39.		
40.	IF phaseStatusGroupPhaseOns.1 ≠ 0x81 OR phaseStatusGroupPhaseOns.2 ≠ 0x00 THEN GOTO Termination	

	Restore *Note:* If it does not go to 1+8 then restore original values and then exit.	
41.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
42.	WHILE phaseStatusGroupPhaseOns.1 = 0x81 AND phaseStatusGroupPhaseOns.2 = 0x00	
43.	DELAY 1 Second	
44.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
45.	WEND *Note:* Wait until a change from 1+8.	
46.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1= 0x02 AND phaseStatusGroupPhaseOns.1= 0x80 *Note:* When it does change, it should be to 2+16.	Pass/Fail
47.	IF phaseStatusGroupPhaseOns.1≠ 0x02 OR phaseStatusGroupPhaseOns.1≠ 0x80 THEN GOTO TerminationRestore *Note:* If it does not go to 2+16 then restore original values and then exit.	
48.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
49.	WHILE phaseStatusGroupPhaseOns.1 = 0x02 AND phaseStatusGroupPhaseOns.2 = 0x80	
50.	DELAY 1 Second	
51.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
52.	WEND *Note:* Wait until a change from 2+16.	
53.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1= 0x12 AND phaseStatusGroupPhaseOns.1 = 0x00 *Note:* When it does change, it should be to 2+5.	Pass/Fail
54.	Set HITL Detector Input 2 = Off	
55.	DELAY .2 Seconds	
56.	IF phaseStatusGroupPhaseOns.1≠ 0x12 OR phaseStatusGroupPhaseOns.1≠ 0x00 THEN GOTO TerminationRestore *Note:* If it does not go to 2+5 then restore original values and then exit.	
Sequence from 3+5 with Calls on 3 and 4 = 4+5 and 3+5		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	

9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	GET ringStatus.1, ringStatus.2	
16.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
17.	DELAY 1 Second	
18.	GET ringStatus.1, ringStatus.2	
19.	WEND *Note:* Wait for 3+5 Green Rest.	
20.	Set HITL Detector Input 4 = On	
21.	DELAY .2 Seconds	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WHILE phaseStatusGroupPhaseOns.1= 0x14 AND phaseStatusGroupPhaseOns.1= 0x00	
24.	DELAY 1 Second	
25.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
26.	WEND *Note:* Wait until a change from 3+5.	
27.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x18 AND phaseStatusGroupPhaseOns.2 = 0x00 *Note:* When it does change, it should change to 4+5.	Pass/Fail
28.	Set HITL Detector Input 4 = Off	
29.	DELAY .2 Seconds	
30.	Set HITL Detector Input 3 = On	
31.	DELAY .2 Seconds	
32.	IF phaseStatusGroupPhaseOns.1 ≠ 0x18 OR phaseStatusGroupPhaseOns.2 ≠ 0x00 THEN GOTO TerminationRestore *Note:* If it does not go to 4+5 then restore original values and then exit.	
33.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
34.	WHILE phaseStatusGroupPhaseOns.1 = 0x18 AND phaseStatusGroupPhaseOns.2 = 0x00	
35.	DELAY 1 Second	
36.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
37.	WEND *Note:* Wait until a change from 4+5.	
38.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x14 AND phaseStatusGroupPhaseOns.2 = 0x00 *Note:* When it does change, it should change to 3+5.	Pass/Fail
39.	Set HITL Detector Input 3 = Off	
40.	DELAY .2 Seconds	

41.	IF phaseStatusGroupPhaseOns.1 ≠ 0x14 OR phaseStatusGroupPhaseOns.2 ≠ 0x00 THEN GOTO TerminationRestore *Note:* If it does not go to 3+5 then restore original values and then exit.	
Sequence from 3+5 with Calls on 3 and 6 = 6+11, 1+6, 1+13, and 3+5		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	GET ringStatus.1, ringStatus.2	
16.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
17.	DELAY 1 Second	
18.	GET ringStatus.1, ringStatus.2	
19.	WEND *Note:* Wait for 3+5 Green Rest.	
20.	Set HITL Detector Input 6 = On	
21.	DELAY .2 Seconds	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WHILE phaseStatusGroupPhaseOns.1= 0x14 AND phaseStatusGroupPhaseOns.2= 0x00	
24.	DELAY 1 Second	
25.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
26.	WEND *Note:* Wait until a change from 3+5.	
27.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x20 AND phaseStatusGroupPhaseOns.2 = 0x04 *Note:* When it does change, it should change to 6+11	Pass/Fail
28.	IF phaseStatusGroupPhaseOns.1 ≠ 0x20 OR phaseStatusGroupPhaseOns.2 ≠ 0x04 THEN GOTO TerminationRestore *Note:* If it does not go to 6+11 then restore original values and then exit.	
29.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	

30.	WHILE phaseStatusGroupPhaseOns.1= 0x20 AND phaseStatusGroupPhaseOns.2= 0x04	
31.	DELAY 1 Second	
32.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
33.	WEND *Note:* Wait until a change from 6+11	
34.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x21 AND phaseStatusGroupPhaseOns.2 = 0x00 *Note:* When it does change, it should change to 1+6.	Pass/Fail
35.	Set HITL Detector Input 6 = Off	
36.	DELAY .2 Seconds	
37.	Set HITL Detector Input 3 = On	
38.	DELAY .2 Seconds	
39.	IF phaseStatusGroupPhaseOns.1 ≠ 0x21 OR phaseStatusGroupPhaseOns.2 ≠ 0x00 THEN GOTO TerminationRestore *Note:* If it does not go to 1+6 then restore original values and then exit.	
40.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
41.	WHILE phaseStatusGroupPhaseOns.1= 0x21 AND phaseStatusGroupPhaseOns.2= 0x00	
42.	DELAY 1 Second	
43.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
44.	WEND *Note:* Wait until a change from 1+6.	
45.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x01 AND phaseStatusGroupPhaseOns.2 = 0x10 *Note:* When it does change, it should change to 1+13.	Pass/Fail
46.	IF phaseStatusGroupPhaseOns.1 ≠ 0x01 OR phaseStatusGroupPhaseOns.2 ≠ 0x10 THEN GOTO TerminationRestore *Note:* If it does not go to 1+13 then restore original values and then exit.	
47.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
48.	WHILE phaseStatusGroupPhaseOns.1= 0x01 AND phaseStatusGroupPhaseOns.2= 0x10	
49.	DELAY 1 Second	
50.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
51.	WEND *Note:* Wait until a change from 1+13.	
52.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x14 AND phaseStatusGroupPhaseOns.2 = 0x00 *Note:* When it does change, it should change to 3+5.	Pass/Fail
53.	Set HITL Detector Input 3 = Off	
54.	DELAY .2 Seconds	
55.	IF phaseStatusGroupPhaseOns.1 ≠ 0x14 OR phaseStatusGroupPhaseOns.2 ≠ 0x00 THEN GOTO	

	TerminationRestore *Note:* If it does not go to 3+5 then restore original values and then exit.	
Sequence from 3+5 with Calls on 3 and 7 = 5+9, 1+7, 1+13, and 3+5		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	GET ringStatus.1, ringStatus.2	
16.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
17.	DELAY 1 Second	
18.	GET ringStatus.1, ringStatus.2	
19.	WEND *Note:* Wait for 3+5 Green Rest.	
20.	Set HITL Detector Input 7 = On	
21.	DELAY .2 Seconds	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WHILE phaseStatusGroupPhaseOns.1= 0x14 AND phaseStatusGroupPhaseOns.2= 0x00	
24.	DELAY 1 Second	
25.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
26.	WEND *Note:* Wait until a change from 3+5.	
27.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x10 AND phaseStatusGroupPhaseOns.2 = 0x01 *Note:* When it does change, it should change to 5+9.	Pass/Fail
28.	IF phaseStatusGroupPhaseOns.1 ≠ 0x10 OR phaseStatusGroupPhaseOns.2 ≠ 0x01 THEN GOTO TerminationRestore *Note:* If it does not go to 5+9 then restore original values and then exit.	
29.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
30.	WHILE phaseStatusGroupPhaseOns.1= 0x10 AND phaseStatusGroupPhaseOns.2= 0x01	

31.	DELAY 1 Second	
32.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
33.	WEND *Note:* Wait until a change from 5+9.	
34.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x41 AND phaseStatusGroupPhaseOns.2 = 0x00 *Note:* When it does change, it should change to 1+7.	Pass/Fail
35.	Set HITL Detector Input 7 = Off	
36.	DELAY .2 Seconds	
37.	Set HITL Detector Input 3 = On	
38.	DELAY .2 Seconds	
39.	IF phaseStatusGroupPhaseOns.1 ≠ 0x41 OR phaseStatusGroupPhaseOns.2 ≠ 0x00 THEN GOTO TerminationRestore *Note:* If it does not go to 1+7 then restore original values and then exit.	
40.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
41.	WHILE phaseStatusGroupPhaseOns.1= 0x41 AND phaseStatusGroupPhaseOns.2= 0x00	
42.	DELAY 1 Second	
43.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
44.	WEND *Note:* Wait until a change from 1+7.	
45.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x01 AND phaseStatusGroupPhaseOns.2 = 0x10 *Note:* When it does change, it should change to 1+13.	Pass/Fail
46.	IF phaseStatusGroupPhaseOns.1 ≠ 0x01 OR phaseStatusGroupPhaseOns.2 ≠ 0x10 THEN GOTO TerminationRestore *Note:* If it does not go to 1+13 then restore original values and then exit.	
47.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
48.	WHILE phaseStatusGroupPhaseOns.1= 0x01 AND phaseStatusGroupPhaseOns.2= 0x10	
49.	DELAY 1 Second	
50.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
51.	WEND *Note:* Wait until a change from 1+13.	
52.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x14 AND phaseStatusGroupPhaseOns.2 = 0x00 *Note:* When it does change, it should change to 3+5.	Pass/Fail
53.	Set HITL Detector Input 3 = Off	
54.	DELAY .2 Seconds	
55.	IF phaseStatusGroupPhaseOns.1 ≠ 0x14 OR phaseStatusGroupPhaseOns.2 ≠ 0x00 THEN GOTO TerminationRestore	

	Note: If it does not go to 3+5 then restore original values and then exit.	
Sequence from 3+5 with Calls on 3 and 8 = 5+9, 1+8, 1+13, and 3+5		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	GET ringStatus.1, ringStatus.2	
16.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
17.	DELAY 1 Second	
18.	GET ringStatus.1, ringStatus.2	
19.	WEND *Note:* Wait for 3+5 Green Rest.	
20.	Set HITL Detector Input 8 = On	
21.	DELAY .2 Seconds	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WHILE phaseStatusGroupPhaseOns.1= 0x14 AND phaseStatusGroupPhaseOns.2= 0x00	
24.	DELAY 1 Second	
25.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
26.	WEND *Note:* Wait until a change from 3+5.	
27.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x10 AND phaseStatusGroupPhaseOns.2 = 0x01 *Note:* When it does change, it should change to 5+9.	Pass/Fail
28.	IF phaseStatusGroupPhaseOns.1 ≠ 0x10 OR phaseStatusGroupPhaseOns.2 ≠ 0x01 THEN GOTO TerminationRestore *Note:* If it does not go to 5+9 then restore original values and then exit.	
29.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
30.	WHILE phaseStatusGroupPhaseOns.1= 0x10 AND phaseStatusGroupPhaseOns.2= 0x01	
31.	DELAY 1 Second	
32.	GET phaseStatusGroupPhaseOns.1,	

	phaseStatusGroupPhaseOns.2	
33.	WEND *Note:* Wait until a change from 5+9.	
34.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x81 AND phaseStatusGroupPhaseOns.2 = 0x00 *Note:* When it does change, it should change to 1+8.	Pass/Fail
35.	Set HITL Detector Input 8 = Off	
36.	DELAY .2 Seconds	
37.	Set HITL Detector Input 3 = On	
38.	DELAY .2 Seconds	
39.	IF phaseStatusGroupPhaseOns.1 ≠ 0x81 OR phaseStatusGroupPhaseOns.2 ≠ 0x00 THEN GOTO TerminationRestore *Note:* If it does not go to 1+8 then restore original values and then exit.	
40.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
41.	WHILE phaseStatusGroupPhaseOns.1= 0x81 AND phaseStatusGroupPhaseOns.2= 0x00	
42.	DELAY 1 Second	
43.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
44.	WEND *Note:* Wait until a change from 1+8.	
45.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x01 AND phaseStatusGroupPhaseOns.2 = 0x10 *Note:* When it does change, it should change to 1+13.	Pass/Fail
46.	IF phaseStatusGroupPhaseOns.1 ≠ 0x01 OR phaseStatusGroupPhaseOns.2 ≠ 0x10 THEN GOTO TerminationRestore *Note:* If it does not go to 1+13 then restore original values and then exit.	
47.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
48.	WHILE phaseStatusGroupPhaseOns.1= 0x01 AND phaseStatusGroupPhaseOns.2= 0x10	
49.	DELAY 1 Second	
50.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
51.	WEND *Note:* Wait until a change from 1+13.	
52.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x14 AND phaseStatusGroupPhaseOns.2 = 0x00 *Note:* When it does change, it should change to 3+5.	Pass/Fail
53.	Set HITL Detector Input 3 = Off	
54.	DELAY .2 Seconds	
55.	IF phaseStatusGroupPhaseOns.1 ≠ 0x14 OR phaseStatusGroupPhaseOns.2 ≠ 0x00 THEN GOTO TerminationRestore *Note:* If it does not go to 3+5 then restore original values and then exit.	

Sequence from 4+5 with Calls on 4 and 6 = 6+12, 1+6, 1+13, and 4+5		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	GET ringStatus.1, ringStatus.2	
16.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
17.	DELAY 1 Second	
18.	GET ringStatus.1, ringStatus.2	
19.	WEND *Note:* Wait for 4+5 Green Rest.	
20.	Set HITL Detector Input 6 = On	
21.	DELAY .2 Seconds	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WHILE phaseStatusGroupPhaseOns.1= 0x18 AND phaseStatusGroupPhaseOns.2= 0x00	
24.	DELAY 1 Second	
25.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
26.	WEND *Note:* Wait until a change from 4+5.	
27.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x20 AND phaseStatusGroupPhaseOns.2 = 0x08 *Note:* When it does change, it should change to 6+12.	Pass/Fail
28.	IF phaseStatusGroupPhaseOns.1 ≠ 0x20 OR phaseStatusGroupPhaseOns.2 ≠ 0x08 THEN GOTO TerminationRestore *Note:* If it does not go to 6+12 then restore original values and then exit.	
29.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
30.	WHILE phaseStatusGroupPhaseOns.1= 0x20 AND phaseStatusGroupPhaseOns.1= 0x08	
31.	DELAY 1 Second	
32.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
33.	WEND	

	Note: Wait until a change from 6+12.	
34.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x21 AND phaseStatusGroupPhaseOns.2 = 0x00 *Note:* When it does change, it should change to 1+6.	Pass/Fail
35.	Set HITL Detector Input 6 = Off	
36.	DELAY .2 Seconds	
37.	Set HITL Detector Input 4 = On	
38.	DELAY .2 Seconds	
39.	IF phaseStatusGroupPhaseOns.1 ≠ 0x21 OR phaseStatusGroupPhaseOns.2 ≠ 0x00 THEN GOTO TerminationRestore *Note:* If it does not go to 1+6 then restore original values and then exit.	
40.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
41.	WHILE phaseStatusGroupPhaseOns.1= 0x21 AND phaseStatusGroupPhaseOns.2= 0x00	
42.	DELAY 1 Second	
43.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
44.	WEND *Note:* Wait until a change from 1+6.	
45.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x01 AND phaseStatusGroupPhaseOns.2 = 0x10 *Note:* When it does change, it should change to 1+13.	Pass/Fail
46.	IF phaseStatusGroupPhaseOns.1 ≠ 0x01 OR phaseStatusGroupPhaseOns.2 ≠ 0x10 THEN GOTO TerminationRestore *Note:* If it does not go to 1+13 then restore original values and then exit.	
47.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
48.	WHILE phaseStatusGroupPhaseOns.1= 0x01 AND phaseStatusGroupPhaseOns.2= 0x10	
49.	DELAY 1 Second	
50.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
51.	WEND *Note:* Wait until a change from 1+13.	
52.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x18 AND phaseStatusGroupPhaseOns.2 = 0x00 *Note:* When it does change, it should change to 4+5.	Pass/Fail
53.	Set HITL Detector Input 4 = Off	
54.	DELAY .2 Seconds	
55.	IF phaseStatusGroupPhaseOns.1 ≠ 0x18 OR phaseStatusGroupPhaseOns.2 ≠ 0x00 THEN GOTO TerminationRestore *Note:* If it does not go to 4+5 then restore original values and then exit.	
Sequence from 4+5 with Calls on 4 and 7 = 5+9, 1+7, 1+13, and 4+5		
1.	GET ringStatus.1, ringStatus.2	

2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	GET ringStatus.1, ringStatus.2	
16.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
17.	DELAY 1 Second	
18.	GET ringStatus.1, ringStatus.2	
19.	WEND *Note:* Wait for 4+5 Green Rest.	
20.	Set HITL Detector Input 7 = On	
21.	DELAY .2 Seconds	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WHILE phaseStatusGroupPhaseOns.1= 0x18 AND phaseStatusGroupPhaseOns.2= 0x00	
24.	DELAY 1 Second	
25.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
26.	WEND *Note:* Wait until a change from 4+5.	
27.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x10 AND phaseStatusGroupPhaseOns.2 = 0x01 *Note:* When it does change, it should change to 5+9.	Pass/Fail
28.	IF phaseStatusGroupPhaseOns.1 ≠ 0x10 OR phaseStatusGroupPhaseOns.2 ≠ 0x01 THEN GOTO TerminationRestore *Note:* If it does not go to 5+9 then restore original values and then exit.	
29.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
30.	WHILE phaseStatusGroupPhaseOns.1= 0x10 AND phaseStatusGroupPhaseOns.2= 0x01	
31.	DELAY 1 Second	
32.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
33.	WEND *Note:* Wait until a change from 5+9.	
34.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 =	Pass/Fail

	0x41 AND phaseStatusGroupPhaseOns.2 = 0x00 *Note:* When it does change, it should change to 1+7.	
35.	Set HITL Detector Input 7 = Off	
36.	DELAY .2 Seconds	
37.	Set HITL Detector Input 4 = On	
38.	DELAY .2 Seconds	
39.	IF phaseStatusGroupPhaseOns.1 ≠ 0x41 OR phaseStatusGroupPhaseOns.2 ≠ 0x00 THEN GOTO TerminationRestore *Note:* If it does not go to 1+7 then restore original values and then exit.	
40.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
41.	WHILE phaseStatusGroupPhaseOns.1= 0x41 AND phaseStatusGroupPhaseOns.2= 0x00	
42.	DELAY 1 Second	
43.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
44.	WEND *Note:* Wait until a change from 1+7.	
45.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x01 AND phaseStatusGroupPhaseOns.2 = 0x10 *Note:* When it does change, it should change to 1+13.	Pass/Fail
46.	IF phaseStatusGroupPhaseOns.1 ≠ 0x01 OR phaseStatusGroupPhaseOns.2 ≠ 0x10 THEN GOTO TerminationRestore *Note:* If it does not go to 1+13 then restore original values and then exit.	
47.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
48.	WHILE phaseStatusGroupPhaseOns.1= 0x01 AND phaseStatusGroupPhaseOns.2= 0x10	
49.	DELAY 1 Second	
50.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
51.	WEND *Note:* Wait until a change from 1+13.	
52.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x18 AND phaseStatusGroupPhaseOns.2 = 0x00 *Note:* When it does change, it should change to 4+5.	Pass/Fail
53.	Set HITL Detector Input 4 = Off	
54.	DELAY .2 Seconds	
55.	IF phaseStatusGroupPhaseOns.1 ≠ 0x18 OR phaseStatusGroupPhaseOns.2 ≠ 0x00 THEN GOTO TerminationRestore *Note:* If it does not go to 4+5 then restore original values and then exit.	
Sequence from 4+5 with Calls on 4 and 8 = 5+9, 1+8, 1+13, and 4+5		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	

4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	GET ringStatus.1, ringStatus.2	
16.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
17.	DELAY 1 Second	
18.	GET ringStatus.1, ringStatus.2	
19.	WEND *Note:* Wait for 4+5 Green Rest.	
20.	Set HITL Detector Input 8 = On	
21.	DELAY .2 Seconds	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WHILE phaseStatusGroupPhaseOns.1= 0x18 AND phaseStatusGroupPhaseOns.2= 0x00	
24.	DELAY 1 Second	
25.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
26.	WEND *Note:* Wait until a change from 4+5.	
27.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x10 AND phaseStatusGroupPhaseOns.2 = 0x01 *Note:* When it does change, it should change to 5+9.	Pass/Fail
28.	IF phaseStatusGroupPhaseOns.1 ≠ 0x10 OR phaseStatusGroupPhaseOns.2 ≠ 0x01 THEN GOTO TerminationRestore *Note:* If it does not go to 5+9 then restore original values and then exit.	
29.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
30.	WHILE phaseStatusGroupPhaseOns.1= 0x10 AND phaseStatusGroupPhaseOns.2= 0x01	
31.	DELAY 1 Second	
32.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
33.	WEND *Note:* Wait until a change from 5+9.	
34.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x81 AND phaseStatusGroupPhaseOns.2 = 0x00	Pass/Fail

	Note: When it does change, it should change to 1+8.	
35.	Set HITL Detector Input 8 = Off	
36.	DELAY .2 Seconds	
37.	Set HITL Detector Input 4 = On	
38.	DELAY .2 Seconds	
39.	IF phaseStatusGroupPhaseOns.1 ≠ 0x81 OR phaseStatusGroupPhaseOns.2 ≠ 0x00 THEN GOTO TerminationRestore *Note:* If it does not go to 1+8 then restore original values and then exit.	
40.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
41.	WHILE phaseStatusGroupPhaseOns.1= 0x81 AND phaseStatusGroupPhaseOns.2= 0x00	
42.	DELAY 1 Second	
43.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
44.	WEND *Note:* Wait until a change from 1+8.	
45.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x01 AND phaseStatusGroupPhaseOns.2 = 0x10 *Note:* When it does change, it should change to 1+13.	Pass/Fail
46.	IF phaseStatusGroupPhaseOns.1 ≠ 0x01 OR phaseStatusGroupPhaseOns.2 ≠ 0x10 THEN GOTO TerminationRestore *Note:* If it does not go to 1+13 then restore original values and then exit.	
47.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
48.	WHILE phaseStatusGroupPhaseOns.1= 0x01 AND phaseStatusGroupPhaseOns.2= 0x10	
49.	DELAY 1 Second	
50.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
51.	WEND *Note:* Wait until a change from 1+13.	
52.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x18 AND phaseStatusGroupPhaseOns.2 = 0x00 *Note:* When it does change, it should change to 4+5.	Pass/Fail
53.	Set HITL Detector Input 4 = Off	
54.	DELAY .2 Seconds	
55.	IF phaseStatusGroupPhaseOns.1 ≠ 0x18 OR phaseStatusGroupPhaseOns.2 ≠ 0x00 THEN GOTO TerminationRestore *Note:* If it does not go to 4+5 then restore original values and then exit.	
Sequence from 1+6 with Calls on 6 and 7 = 1+7 and 1+6		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND	

	Note: Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 6 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+6.	
13.	Set HITL Detector Input 6 = Off	
14.	DELAY .2 Seconds	
15.	GET ringStatus.1, ringStatus.2	
16.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
17.	DELAY 1 Second	
18.	GET ringStatus.1, ringStatus.2	
19.	WEND *Note:* Wait for 1+6 Green Rest.	
20.	Set HITL Detector Input 7 = On	
21.	DELAY .2 Seconds	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WHILE phaseStatusGroupPhaseOns.1= 0x21 AND phaseStatusGroupPhaseOns.1= 0x00	
24.	DELAY 1 Second	
25.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
26.	WEND *Note:* Wait until a change from 1+6.	
27.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x41 AND phaseStatusGroupPhaseOns.2 = 0x00 *Note:* When it does change, it should change to 1+7.	Pass/Fail
28.	Set HITL Detector Input 7 = Off	
29.	DELAY .2 Seconds	
30.	Set HITL Detector Input 6 = On	
31.	DELAY .2 Seconds	
32.	IF phaseStatusGroupPhaseOns.1 ≠ 0x41 OR phaseStatusGroupPhaseOns.2 ≠ 0x00 THEN GOTO TerminationRestore *Note:* If it does not go to 1+7 then restore original values and then exit.	
33.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
34.	WHILE phaseStatusGroupPhaseOns.1= 0x41 AND phaseStatusGroupPhaseOns.2= 0x00	
35.	DELAY 1 Second	
36.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
37.	WEND *Note:* Wait until a change from 1+7.	
38.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 =	Pass/Fail

	0x21 AND phaseStatusGroupPhaseOns.2 = 0x00 *Note:* When it does change, it should change to 1+6.	
39.	Set HITL Detector Input 6 = Off	
40.	DELAY .2 Seconds	
41.	IF phaseStatusGroupPhaseOns.1 ≠ 0x21 OR phaseStatusGroupPhaseOns.2 ≠ 0x00 THEN GOTO TerminationRestore *Note:* If it does not go to 1+6 then restore original values and then exit.	
Sequence from 1+6 with Calls on 6 and 8 = 1+8 and 1+6		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 6 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+6.	
13.	Set HITL Detector Input 6 = Off	
14.	DELAY .2 Seconds	
15.	GET ringStatus.1, ringStatus.2	
16.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
17.	DELAY 1 Second	
18.	GET ringStatus.1, ringStatus.2	
19.	WEND *Note:* Wait for 1+6 Green Rest.	
20.	Set HITL Detector Input 8 = On	
21.	DELAY .2 Seconds	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WHILE phaseStatusGroupPhaseOns.1= 0x21 AND phaseStatusGroupPhaseOns.2= 0x00	
24.	DELAY 1 Second	
25.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
26.	WEND *Note:* Wait until a change from 1+6.	
27.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x81 AND phaseStatusGroupPhaseOns.2 = 0x00 *Note:* When it does change, it should change to 1+8.	Pass/Fail
28.	Set HITL Detector Input 8 = Off	
29.	DELAY .2 Seconds	

30.	Set HITL Detector Input 6 = On	
31.	DELAY .2 Seconds	
32.	IF phaseStatusGroupPhaseOns.1 ≠ 0x81 OR phaseStatusGroupPhaseOns.2 ≠ 0x00 THEN GOTO TerminationRestore *Note:* If it does not go to 1+8 then restore original values and then exit.	
33.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
34.	WHILE phaseStatusGroupPhaseOns.1= 0x81 AND phaseStatusGroupPhaseOns.2= 0x00	
35.	DELAY 1 Second	
36.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
37.	WEND *Note:* Wait until a change from 1+8.	
38.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x21 AND phaseStatusGroupPhaseOns.2 = 0x00 *Note:* When it does change, it should change to 1+6.	Pass/Fail
39.	Set HITL Detector Input 6 = Off	
40.	DELAY .2 Seconds	
41.	IF phaseStatusGroupPhaseOns.1 ≠ 0x21 OR phaseStatusGroupPhaseOns.2 ≠ 0x00 THEN GOTO TerminationRestore *Note:* If it does not go to 1+6 then restore original values and then exit.	
Sequence from 1+7 with Calls on 7 and 8 = 1+8 and 1+7		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	GET ringStatus.1, ringStatus.2	
16.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
17.	DELAY 1 Second	
18.	GET ringStatus.1, ringStatus.2	
19.	WEND	

	Note: Wait for 1+7 Green Rest.	
20.	Set HITL Detector Input 8 = On	
21.	DELAY .2 Seconds	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WHILE phaseStatusGroupPhaseOns.1= 0x41 AND phaseStatusGroupPhaseOns.2= 0x00	
24.	DELAY 1 Second	
25.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
26.	WEND *Note:* Wait until a change from 1+ 7.	
27.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x81 AND phaseStatusGroupPhaseOns.2 = 0x00 *Note:* When it does change, it should change to 1+8.	Pass/Fail
28.	Set HITL Detector Input 8 = Off	
29.	DELAY .2 Seconds	
30.	Set HITL Detector Input 7 = On	
31.	DELAY .2 Seconds	
32.	IF phaseStatusGroupPhaseOns.1 ≠ 0x81 OR phaseStatusGroupPhaseOns.2 ≠ 0x00 THEN GOTO TerminationRestore *Note:* If it does not go to 1+8 then restore original values and then exit.	
33.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
34.	WHILE phaseStatusGroupPhaseOns.1= 0x81 AND phaseStatusGroupPhaseOns.2= 0x00	
35.	DELAY 1 Second	
36.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
37.	WEND *Note:* Wait until a change from 1+ 8.	
38.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 = 0x41 AND phaseStatusGroupPhaseOns.2 = 0x00 *Note:* When it does change, it should change to 1+7.	Pass/Fail
39.	Set HITL Detector Input 7 = Off	
40.	DELAY .2 Seconds	
41.	IF phaseStatusGroupPhaseOns.1 ≠ 0x41 OR phaseStatusGroupPhaseOns.2 ≠ 0x00 THEN GOTO TerminationRestore *Note:* If it does not go to 1+7 then restore original values and then exit.	
Termination Restore		
1. (Termination Restore)	FOR Phase = 1 TO 16	
2.	SET phaseMinimumGreen.Phase = [currentMinGrn.Phase], phasePassage.Phase = [currentPassage.Phase], and phaseMaximum1.Phase = [currentMax1.Phase]	
3.	NEXT Phase	
4.	Set HITL Detector Input 2, 3, 4, 6, 7, and 8 = Off	
5.	DELAY .2 Seconds	

Test Case Results				
Tested By:		Date Tested		Pass/Fail
Test Case Notes:	<notes>			
Version History:	v1.00 04/07/06 Initial draft – RDR v1.01 07/05/06 Updated notes – RDR v1.02 07/24/06 Implemented script and proofed – JJ			

Four-Phase Diamond Detector Operations

The test cases check the operation of detector inputs 1 through 18 when a traffic signal controller is configured for four-phase diamond operation. The test cases are defined upon the Detector Operation Requirements appearing on page 18 of TxDOT DMS-11170, Fully Actuated, Solid-State Traffic Signal Controller Assembly – dated August 2004 (4).

Detector 1 Operations

Test Case: **TC001**	*Title*:	**Detector 1 Operations**	
	Description:	Verifies the operation of Detector 1 to call Phase 6 if overlap A is not green and there is no call on phase 7 or 8, and extends intervals 2516B, 2517B, 2518B, 4517B, 4518B, 1517B, and 3518B.	
	Constants:		
	Variables:		
	Pass/Fail Criteria:	The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*		*Results*
Setup			
	PERFORM Detector Operations Setup – TC019 if not already done so.		
Detector 1 No Call on Phase 6 when Overlap A is Green			
1.	GET ringStatus.1, ringStatus.2		
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03		
3.	DELAY 1 Second		
4.	GET ringStatus.1, ringStatus.2		
5.	WEND *Note:* Loop until controller rests somewhere.		
6.	Set HITL Detector Input 6 = On		
7.	DELAY .2 Seconds		
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2		
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0X00		
10.	DELAY 1 Second		
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2		
12.	WEND *Note:* Wait until controller reaches 1+6.		
13.	Set HITL Detector Input 6 = Off		
14.	DELAY .2 Seconds		
15.	GET ringStatus.1, ringStatus.2		
16.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03		
17.	DELAY 1 Second		
18.	GET ringStatus.1, ringStatus.2		

19.	WEND *Note:* Wait for 1+6 Green Rest.	
20.	GET overlapStatusGroupGreens.1 *Note:* overlapStatusGroupGreens is optional and a GET may return a noSuchName. This also assumes that Overlap A = 1+2+X.	
21.	IF RESPONSE ERROR = noError THEN	
22.	VERIFY that RESPONSE VALUE overlapStatusGroupGreens.1 AND 0x01 = 0x01 *Note:* Verifies that Overlap A = Green.	Pass/Fail
23.	ENDIF	
24.	Set HITL Detector Input 1 = On	
25.	DELAY3 Seconds (3 full seconds)	
26.	GET phaseStatusGroupVehCalls.1	
27.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 AND 0x20 = 0x00 *Note:* Ensure that Phase 6 is not registering a Vehicle Call.	Pass/Fail
28.	Set HITL Detector Input 1 = Off	
29.	DELAY .2 Seconds	
Detector 1 No Call on Phase 6 when Overlap A is not Green and a Call on Phase 7		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	GET overlapStatusGroupGreens.1 *Note:* overlapStatusGroupGreens is optional and a GET may return a noSuchName. This also assumes that Overlap A = 1+2+X.	
16.	IF RESPONSE ERROR = noError THEN	
17.	VERIFY that RESPONSE VALUE overlapStatusGroupGreens.1 AND 0x01 = 0x00 *Note:* Verifies that Overlap A = NOT Green	Pass/Fail
18.	ENDIF	
19.	Set HITL Detector Input 4 = On	
20.	DELAY .2 Seconds	

21.	Set HITL Detector Input 7 = On	
22.	DELAY 2 Seconds (2 full seconds)	
23.	Set HITL Detector Input 1 = On	
24.	DELAY 2 Seconds (2 full seconds)	
25.	GET phaseStatusGroupVehCalls.1	
26.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 AND 0x20 = 0x00 *Note:* Ensure that Phase 6 is not registering a Vehicle Call.	Pass/Fail
27.	Set HITL Detector Input 1 = Off, 7 = Off, and 4 = Off	
28.	DELAY .2 Seconds	
Detector 1 No Call on Phase 6 when Overlap A is not Green and a Call on Phase 8		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	GET overlapStatusGroupGreens.1 *Note:* overlapStatusGroupGreens is optional and a GET may return a noSuchName. This also assumes that Overlap A = 1+2+X.	
16.	IF RESPONSE ERROR = noError THEN	
17.	VERIFY that RESPONSE VALUE overlapStatusGroupGreens.1 AND 0x01 = 0x00 *Note:* Verifies that Overlap A = NOT Green	Pass/Fail
18.	ENDIF	
19.	Set HITL Detector Input 4 = On	
20.	DELAY .2 Seconds	
21.	Set HITL Detector Input 8 = On	
22.	DELAY 2 Seconds (2 full seconds)	
23.	Set HITL Detector Input 1 = On	
24.	DELAY 2 Seconds	
25.	GET phaseStatusGroupVehCalls.1	
26.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 AND 0x20 = 0x00 *Note:* Ensure that Phase 6 is not registering a Vehicle Call.	Pass/Fail
27.	Set HITL Detector Input 1 = Off, 8 = Off, and 4 = Off	
28.	DELAY .2 Seconds	

Detector 1 calls Phase 6 when Overlap A is not Green and no Calls on Phase 7 or Phase 8		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	GET overlapStatusGroupGreens.1 *Note:* overlapStatusGroupGreens is optional and a GET may return a noSuchName. This also assumes that Overlap A = 1+2+X.	
16.	IF RESPONSE ERROR = noError THEN	
17.	VERIFY that RESPONSE VALUE overlapStatusGroupGreens.1 AND 0x01 = 0x00 *Note:* Verifies that Overlap A = NOT Green	Pass/Fail
18.	ENDIF	
19.	Set HITL Detector Input 1 = On	
20.	DELAY 2 Seconds (2 full seconds)	
21.	GET phaseStatusGroupVehCalls.1	
22.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 AND 0x20 = 0x20 *Note:* Check for call on Phase 6.	Pass/Fail
23.	Set HITL Detector Input 1 = Off	
24.	DELAY .2 Seconds	
Detector 1 extends Phase 9 when Phase 9 is Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1,	

	phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 7 = On	
16.	DELAY .2 Seconds	
17.	Set HITL Detector Input 1 = On	
18.	DELAY .2 Seconds	
19.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
20.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x10 AND phaseStatusGroupPhaseOns.2 ≠ 0X01	
21.	DELAY 1 Second	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WEND *Note:* Wait until controller reaches 5+9.	
24.	GET ringStatus.1	
25.	WHILE ringStatus.1 AND 0x10 ≠ 0x10 (xxx1xxxx = maxout)	
26.	DELAY 1 Second	
27.	GET ringStatus.1	
28.	WEND *Note:* Wait for Max Out Indication. Ring 1 = 2, 3, 4, ***9***, 11, & 12 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
29.	GET phaseStatusGroupPhaseOns.2	
30.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.2 AND 0x01 = 0x01 *Note:* Ensure that Max Out occurred on Phase 9.	Pass/Fail
31.	Set HITL Detector Input 1 = Off and 7 = Off	
32.	DELAY .2 Seconds	
Teardown		
	PERFORM Detector Teardown – TC020 if not proceeding to another detector operation test case	

Test Case Results				
Tested By:		Date Tested		Pass/Fail
Test Case Notes:	<notes>			
Version History:	v1.00 04/12/06 Initial Draft – RDR v1.01 07/05/06 Updated notes – RDR v1.02 07/20/06 Implemented script and proofed – JJ			

Detector 2 Operations

Test Case: **TC002**	Title:	**Detector 2 Operations**	
	Description:	Verifies the operation of Detector 2 to call and extend Phase 2	
	Constants:		
	Variables:	currentRedClear currentYellowChange	
	Pass/Fail Criteria:	The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*		*Results*
Setup			
	PERFORM Detector Operations Setup – TC019 if not already done so.		
Detector 2 calls Phase 2 during 1+6 Green			
1.	GET ringStatus.1, ringStatus.2		
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03		
3.	DELAY 1 Second		
4.	GET ringStatus.1, ringStatus.2		
5.	WEND *Note:* Loop until controller rests in green somewhere.		
6.	Set HITL Detector Input 6 = On		
7.	DELAY .2 Seconds		
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2		
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0x00		
10.	DELAY 1 Second		
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2		
12.	WEND *Note:* Wait until controller reaches 1+6.		
13.	Set HITL Detector Input 6 = Off		
14.	DELAY .2 Seconds		
15.	Set HITL Detector Input 6 = On		
16.	DELAY 2 Seconds (2 full seconds)		
17.	GET phaseStatusGroupVehCalls.1		
18.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 AND 0x02 = 0x00 *Note:* Ensure that Phase 2 is NOT registering a Vehicle Call during 6 Green without a Detector 2 Call.		Pass/Fail
19.	Set HITL Detector Input 2 = On		
20.	DELAY 2 Seconds (2 full seconds)		
21.	GET phaseStatusGroupVehCalls.1		
22.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 AND 0x02 = 0x02 *Note:* Ensure that Phase 2 is registering a Vehicle Call during the Green of Phase 6 with a Detector 2 Call.		Pass/Fail
23.	Set HITL Detector Input 6 and Input 2 = Off		
24.	DELAY 2 Seconds (2 full seconds)		

25.	GET phaseStatusGroupVehCalls.1	
26.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 AND 0x02 = 0x00 *Note:* Ensure that Phase 2 is NOT registering a Vehicle Call after Detector 2 Call is removed.	Pass/Fail
27.	GET phaseRedClear.6 = [currentRedClear] and phaseYellowChange.6 = [currentYellowChange]	
28.	SET phaseRedClear.6 = 70 and phaseYellowChange.6 = 70 *Note*: Increase Red Clear and Yellow Change time to ensure enough time for vehicle calls to register on controller during those phases.	
29.	Set HITL Detector Input 7 = On *Note:* This causes 1+6 to advance to Yellow.	
30.	DELAY 2 Seconds	
31.	Set HITL Detector Input 7 = Off	
32.	DELAY .2 Seconds	
33.	GET ringStatus.2	
34.	WHILE ringStatus.2 AND 0x07 ≠ 0x04	
35.	DELAY 1 Second	
36.	GET ringStatus.2	
37.	WEND *Note:* Wait for 6 Yellow. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, ***6***, 7, 8, & 13	
38.	GET phaseStatusGroupVehCalls.1	
39.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 AND 0x02 = 0x00 *Note:* Ensure that Phase 2 is NOT registering a Vehicle Call during Phase 6 Yellow without a Detector 2 Call.	Pass/Fail
40.	Set HITL Detector Input 2 = On	
41.	DELAY 2 Seconds (Give it some time to make sure it is registered)	
42.	GET phaseStatusGroupVehCalls.1	
43.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 AND 0x02 = 0x02 *Note:* Ensure that Phase 2 is registering a Vehicle Call during Phase 6 Yellow with a Detector 2 Call.	Pass/Fail
44.	Set HITL Detector Input 2 = Off	
45.	DELAY 2 Seconds (Give it some time to make sure it is cleared)	
46.	GET phaseStatusGroupVehCalls.1	
47.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 AND 0x02 = 0x00 *Note:* Ensure that Phase 2 is NOT registering a Vehicle Call after Detector 2 Call is removed.	Pass/Fail
48.	GET ringStatus.2	
49.	WHILE ringStatus.2 AND 0x07 ≠ 0x05	
50.	DELAY 1 Second	
51.	GET ringStatus.2	

52.	WEND *Note:* Wait for 6 Red (CBS = xxxxx101). Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, ***6***, 7, 8, & 13	
53.	GET phaseStatusGroupVehCalls.1	
54.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 AND 0x02 = 0x00 *Note:* Ensure that Phase 2 is NOT registering a Vehicle Call during Phase 6 Red without a Detector 2 Call.	Pass/Fail
55.	Set HITL Detector Input 2 = On	
56.	DELAY 2 Seconds (Give it some time to make sure it is registered)	
57.	GET phaseStatusGroupVehCalls.1	
58.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 AND 0x02 = 0x02 *Note:* Ensure that Phase 2 is registering a Vehicle Call during the Phase 6 Red with a Detector 2 Call.	Pass/Fail
59.	Set HITL Detector Input 2 = Off	
60.	DELAY 2 Seconds (Give it some time to make sure it is cleared)	
61.	GET phaseStatusGroupVehCalls.1	
62.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 AND 0x02 = 0x00 *Note:* Ensure that Phase 2 is NOT registering a Vehicle Call after the Detector 2 Call is removed.	Pass/Fail
63.	SET phaseRedClear.6 = [currentRedClear] and phaseYellowChange.6 = [currentYellowChange]	
Detector 2 calls Phase 2 during 1+7 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches or is in 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 2 = On	
16.	DELAY 2 Seconds (2 full seconds)	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	

18.	WHILE phaseStatusGroupGreens.1 = 0x41 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x02 ≠ 0x02	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x41 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x02 = 0x02 *Note*: Ensure that Phase 2 registers a Vehicle Call during 1+7 green.	Pass/Fail
23.	Set HITL Detector Input 2 = Off	
24.	DELAY .2 Seconds	
Detector 2 calls Phase 2 during 1+8 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 2 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x81 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x02 ≠ 0x02	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x81 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x02 = 0x02 *Note:* Ensure that Phase 2 registers a Vehicle Call during 1+8 Green.	Pass/Fail
23.	Set HITL Detector Input 2 = Off	
24.	DELAY .2 Seconds	
Detector 2 calls Phase 2 during 1+13 Green		
1.	GET ringStatus.1, ringStatus.2	

2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 3 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x01 AND phaseStatusGroupPhaseOns.2 ≠ 0x10	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 1+13.	
22.	Set HITL Detector Input 3 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 2 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x01 AND phaseStatusGroupGreens.2 = 0x10 AND phaseStatusGroupVehCalls.1 AND 0x02 ≠ 0x02	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x01 AND phaseStatusGroupGreens.2 = 0x10 AND phaseStatusGroupVehCalls.1 AND 0x02 = 0x02 *Note:* Ensure that Phase 2 registers a Vehicle Call during 1+13 Green.	Pass/Fail
32.	Set HITL Detector Input 2 = Off	
33.	DELAY .2 Seconds	
Detector 2 calls Phase 2 during 3+5 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	

5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 2 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x14 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x02 ≠ 0x02	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x14 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x02 = 0x02 *Note:* Ensure that Phase 2 registers a Vehicle Call during 3+5 Green.	Pass/Fail
23.	Set HITL Detector Input 2 = Off	
24.	DELAY .2 Seconds	
Detector 2 calls Phase 2 during 4+5 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 2 = On	

16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x18 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x02 ≠ 0x02	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x18 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x02 = 0x02 *Note:* Ensure that Phase 2 registers a Vehicle Call during 4+5 Green.	Pass/Fail
23.	Set HITL Detector Input 2 = Off	
24.	DELAY .2 Seconds	
Detector 2 calls Phase 2 during 6+12 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 6 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x20 AND phaseStatusGroupPhaseOns.2 ≠ 0x08	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 6+12.	
22.	Set HITL Detector Input 6 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 2 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	

27.	WHILE phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x08 AND phaseStatusGroupVehCalls.1 AND 0x02 ≠ 0x02	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x08 AND phaseStatusGroupVehCalls.1 AND 0x02 = 0x02 *Note:* Ensure that Phase 2 registers a Vehicle Call during 6+12 Green.	Pass/Fail
32.	Set HITL Detector Input 2 = Off	
33.	DELAY .2 Seconds	
Detector 2 calls Phase 2 during 6+11 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 6 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x20 AND phaseStatusGroupPhaseOns.2 ≠ 0x04	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 6+11.	
22.	Set HITL Detector Input 6 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 2 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x04 AND phaseStatusGroupVehCalls.1 AND 0x02 ≠ 0x02	

28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x04 AND phaseStatusGroupVehCalls.1 AND 0x02 = 0x02 *Note:* Ensure that Phase 2 registers a Vehicle Call during 6+11 Green.	Pass/Fail
32.	Set HITL Detector Input 2 = Off	
33.	DELAY .2 Seconds	
Detector 2 calls Phase 2 during 5+9 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 7 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x10 AND phaseStatusGroupPhaseOns.2 ≠ 0x01	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 5+9.	
22.	Set HITL Detector Input 7 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 2 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x10 AND phaseStatusGroupGreens.2 = 0x01 AND phaseStatusGroupVehCalls.1 AND 0x02 ≠ 0x02	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	

30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x10 AND phaseStatusGroupGreens.2 = 0x01 AND phaseStatusGroupVehCalls.1 AND 0x02 = 0x02 *Note:* Ensure that Phase 2 registers a Vehicle Call during 5+9 Green.	Pass/Fail
32.	Set HITL Detector Input 2 = Off	
33.	DELAY .2 Seconds	
Detector 2 extends Phase 2 when Phase 2 is Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 2 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x12 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 2+5.	
13.	Set HITL Detector Input 6 = On	
14.	DELAY .2 Seconds	
15.	GET ringStatus.1	
16.	WHILE ringStatus.1 AND 0x10 ≠ 0x10 (xxx1xxxx = maxout)	
17.	DELAY 1 Second	
18.	GET ringStatus.1	
19.	WEND *Note:* Wait for Max Out Indication. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
20.	GET phaseStatusGroupPhaseOns.1	
21.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x02 = 0x02 *Note:* Ensure that Max Out occurred on Phase 2.	Pass/Fail
22.	Set HITL Detector Input 2 = Off and 6 = Off	
23.	DELAY .2 Seconds	

Teardown		
	PERFORM Detector Teardown – TC020 if not proceeding to another detector operation test case	

Test Case Results				
Tested By:		Date Tested		Pass/Fail
Test Case Notes:	<notes>			
Version History:	v1.00 04/12/06 Initial Draft (Calls during Yellow and Red are only tested for Phase 6) – RDR v1.01 07/05/06 Updated notes – RDR v1.02 07/20/06 Implemented script and proofed – JJ			

Detector 3 Operations

Test Case: **TC003**	*Title*: Description: Constants: Variables: Pass/Fail Criteria:	**Detector 3 Operations** Verifies the operation of Detector 3 to call and extend Phase 3 under specific conditions and to extend interval 3516B. The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.

Test Step Number	*Test Procedure*	*Results*
Setup		
	PERFORM Detector Operations Setup – TC019 if not already done so.	
Detector 3 calls Phase 3 when Phase 3 and Phase 11 are not Green (In 1+6 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 6 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+6.	
13.	Set HITL Detector Input 6 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 3 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	

18.	WHILE phaseStatusGroupGreens.1 = 0x21 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x04 ≠ 0x04	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x21 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x04 = 0x04 *Note*: Ensure that Phase 3 registers a Vehicle Call during 1+6 Green.	Pass/Fail
23.	Set HITL Detector Input 3 = Off	
24.	DELAY .2 Seconds	
Detector 3 calls Phase 3 when Phase 3 and Phase 11 are not Green (In 1+7 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 3 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x41 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x04 ≠ 0x04	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x41 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x04 = 0x04 *Note:* Ensure that Phase 3 registers a Vehicle Call during 1+7 Green.	Pass/Fail
23.	Set HITL Detector Input 3 = Off	
24.	DELAY .2 Seconds	

Detector 3 calls Phase 3 when Phase 3 and Phase 11 are not Green (In 1+8 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 3 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x81 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x04 ≠ 0x04	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x81 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x04 = 0x04 *Note*: Ensure that Phase 3 registers a Vehicle Call during 1+8 green.	Pass/Fail
23.	Set HITL Detector Input 3 = Off	
24.	DELAY .2 Seconds	
Detector 3 calls Phase 3 when Phase 3 and Phase 11 are not Green (In 2+16 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	

11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 2 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x02 AND phaseStatusGroupPhaseOns.2 ≠ 0x80	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 2+16.	
22.	Set HITL Detector Input 2 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 3 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x80 AND phaseStatusGroupVehCalls.1 AND 0x04 ≠ 0x04	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x80 AND phaseStatusGroupVehCalls.1 AND 0x04 = 0x04 *Note*: Ensure that Phase 3 registers a Vehicle Call during 2+16 green.	Pass/Fail
32.	Set HITL Detector Input 3 = Off	
33.	DELAY .2 Seconds	
Detector 3 calls Phase 3 when Phase 3 and Phase 11 are not Green (In 2+15 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	

11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 2 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x02 AND phaseStatusGroupPhaseOns.2 ≠ 0x40	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 2+15.	
22.	Set HITL Detector Input 2 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 3 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x40 AND phaseStatusGroupVehCalls.1 AND 0x04 ≠ 0x04	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x40 AND phaseStatusGroupVehCalls.1 AND 0x04 = 0x04 *Note*: Ensure that Phase 3 registers a Vehicle Call during 2+15 Green.	Pass/Fail
32.	Set HITL Detector Input 3 = Off	
33.	DELAY .2 Seconds	
Detector 3 calls Phase 3 when Phase 3 and Phase 11 are not Green (In 1+13 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	

11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 2 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x02 AND phaseStatusGroupPhaseOns.2 ≠ 0x40	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 2+15.	
22.	Set HITL Detector Input 2 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 3 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x40 AND phaseStatusGroupVehCalls.1 AND 0x04 ≠ 0x04	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x40 AND phaseStatusGroupVehCalls.1 AND 0x04 = 0x04 *Note*: Ensure that Phase 3 registers a Vehicle Call during 2+15 green.	Pass/Fail
32.	Set HITL Detector Input 3 = Off	
33.	DELAY .2 Seconds	
Detector 3 calls Phase 3 when Phase 3 and Phase 11 are not Green (In 2+5 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 2 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x12 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	

11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 2+5.	
13.	Set HITL Detector Input 2 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 3 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x12 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x04 ≠ 0x04	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x12 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x04 = 0x04 *Note*: Ensure that Phase 3 registers a Vehicle Call during 2+5 Green.	
23.	Set HITL Detector Input 3 = Off	Pass/Fail
24.	DELAY .2 Seconds	
Detector 3 calls Phase 3 when Phase 3 and Phase 11 are not Green (In 4+5 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 3 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x18 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x04 ≠ 0x04	
19.	DELAY 1 Second	

20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x18 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x04 = 0x04 *Note*: Ensure that Phase 3 registers a Vehicle Call during 4+5 Green.	Pass/Fail
23.	Set HITL Detector Input 3 = Off	
24.	DELAY .2 Seconds	
Detector 3 calls Phase 3 when Phase 3 and Phase 11 are not Green (In 6+12 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 6 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x20 AND phaseStatusGroupPhaseOns.2 ≠ 0x08	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 6+12.	
22.	Set HITL Detector Input 6 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 3 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x08 AND phaseStatusGroupVehCalls.1 AND 0x04 ≠ 0x04	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	

30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x08 AND phaseStatusGroupVehCalls.1 AND 0x04 = 0x04 *Note*: Ensure that Phase 3 registers a Vehicle Call during 6+12 Green.	Pass/Fail
32.	Set HITL Detector Input 3 = Off	
33.	DELAY .2 Seconds	
Detector 3 calls Phase 3 when Phase 3 and Phase 11 are not Green (In 5+9 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 7 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x10 AND phaseStatusGroupPhaseOns.2 ≠ 0x01	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 5+9.	
22.	Set HITL Detector Input 7 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 3 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x10 AND phaseStatusGroupGreens.2 = 0x01 AND phaseStatusGroupVehCalls.1 AND 0x04 ≠ 0x04	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	

31.	VERIFY phaseStatusGroupGreens.1 = 0x10 AND phaseStatusGroupGreens.2 = 0x01 AND phaseStatusGroupVehCalls.1 AND 0x04 = 0x04 *Note*: Ensure that Phase 3 registers a Vehicle Call during 5+9 Green.	
32.	Set HITL Detector Input 3 = Off	
33.	DELAY .2 Seconds	
Detector 3 extends Phase 3 during Phase 3		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 6 = On	
14.	DELAY .2 Seconds	
15.	GET ringStatus.1	
16.	WHILE ringStatus.1 AND 0x10 ≠= 0x10 (xxx1xxxx = maxout)	
17.	DELAY 1 Second	
18.	GET ringStatus.1	
19.	WEND *Note:* Wait for Max Out Indication. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
20.	GET phaseStatusGroupPhaseOns.1	
21.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x04 = 0x04 *Note:* Ensure that Max Out occurred on Phase 3	
22.	Set HITL Detector Input 3 = Off and 6 = Off	
23.	DELAY .2 Seconds	
Detector 3 extends Phase 11 during Phase 11		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	

8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 6 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x20 AND phaseStatusGroupPhaseOns.2 ≠ 0x04	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 6+11.	
22.	Set HITL Detector Input 6 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 3 = On	
25.	DELAY .2 Seconds	
26.	Set HITL Detector Input 4 = On	
27.	DELAY .2 Seconds	
28.	GET ringStatus.1	
29.	WHILE ringStatus.1 AND 0x10 ≠ 0x10 (xxx1xxxx = maxout)	
30.	DELAY 1 Second	
31.	GET ringStatus.1	
32.	WEND *Note:* Wait for Max Out Indication. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
33.	GET phaseStatusGroupPhaseOns.2	
34.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.2 AND 0x04 = 0x04 *Note:* Ensure that Max Out occurred on Phase 11	
35.	Set HITL Detector Input 3 = Off and 4 = Off	
36.	DELAY .2 Seconds	
Teardown		
	PERFORM Detector Teardown – TC020 if not proceeding to another detector operation test case	

Test Case Results				
Tested By:		Date Tested		Pass/Fail
Test Case Notes:	<notes>			
Version History:	v1.00 04/14/06 Initial Draft – RDR v1.01 07/05/06 Updated notes – RDR v1.02 07/19/06 Implemented script and proofed – JJ			

Detector 4 Operations

Test Case: **TC004**	Title:	**Detector 4 Operations**	
	Description:	Verifies the operation of Detector 4 to call and extend Phase 4 under specific conditions and to extend interval 4516B.	
	Constants:		
	Variables:		
	Pass/Fail Criteria:	The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*		*Results*
Setup			
	PERFORM Detector Operations Setup – TC019 if not already done so.		
Detector 4 calls Phase 4 when Phase 4 and Phase 12 are not Green (In 1+6 Green)			
1.	GET ringStatus.1, ringStatus.2		
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03		
3.	DELAY 1 Second		
4.	GET ringStatus.1, ringStatus.2		
5.	WEND *Note:* Loop until controller rests in green somewhere.		
6.	Set HITL Detector Input 6 = On		
7.	DELAY .2 Seconds		
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2		
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0x00		
10.	DELAY 1 Second		
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2		
12.	WEND *Note:* Wait until controller reaches 1+6.		
13.	Set HITL Detector Input 6 = Off		
14.	DELAY .2 Seconds		
15.	Set HITL Detector Input 4 = On		
16.	DELAY .2 Seconds		
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1		
18.	WHILE phaseStatusGroupGreens.1 = 0x21 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x08 ≠ 0x08		
19.	DELAY 1 Second		
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1		
21.	WEND		
22.	VERIFY phaseStatusGroupGreens.1 = 0x21 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x08 = 0x08 *Note*: Ensure that Phase 4 registers a Vehicle Call during 1+6 Green.		Pass/Fail
23.	Set HITL Detector Input 4 = Off		
24.	DELAY .2 Seconds		

Detector 4 calls Phase 4 when Phase 4 and Phase 12 are not Green (In 1+7 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 4 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x41 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x08 ≠ 0x08	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x41 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x08 = 0x08 *Note*: Ensure that Phase 4 registers a Vehicle Call during 1+7 Green.	Pass/Fail
23.	Set HITL Detector Input 4 = Off	
24.	DELAY .2 Seconds	
Detector 4 calls Phase 4 when Phase 4 and Phase 12 are not Green (In 1+8 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	

11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 4 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x81 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x08 ≠ 0x08	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x81 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x08 = 0x08 *Note*: Ensure that Phase 4 registers a Vehicle Call during 1+8 Green.	Pass/Fail
23.	Set HITL Detector Input 4 = Off	
24.	DELAY .2 Seconds	
Detector 4 calls Phase 4 when Phase 4 and Phase 12 are not Green (In 2+16 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 2 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x02 AND phaseStatusGroupPhaseOns.2 ≠ 0x80	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	

21.	WEND *Note:* Wait until controller reaches 2+16.	
22.	Set HITL Detector Input 2 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 4 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x80 AND phaseStatusGroupVehCalls.1 AND 0x08 ≠ 0x08	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x80 AND phaseStatusGroupVehCalls.1 AND 0x08 = 0x08 *Note*: Ensure that Phase 4 registers a Vehicle Call during 2+16. Green.	Pass/Fail
32.	Set HITL Detector Input 4 = Off	
33.	DELAY .2 Seconds	
Detector 4 calls Phase 4 when Phase 4 and Phase 12 are not Green (In 2+15 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 2 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x02 AND phaseStatusGroupPhaseOns.2 ≠ 0x40	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	

21.	WEND *Note:* Wait until controller reaches 2+15.	
22.	Set HITL Detector Input 2 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 4 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x40 AND phaseStatusGroupVehCalls.1 AND 0x08 ≠ 0x08	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x40 AND phaseStatusGroupVehCalls.1 AND 0x08 = 0x08 *Note*: Ensure that Phase 4 registers a Vehicle Call during 2+15 Green.	Pass/Fail
32.	Set HITL Detector Input 4 = Off	
33.	DELAY .2 Seconds	
Detector 4 calls Phase 4 when Phase 4 and Phase 12 are not Green (In 1+13 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 6 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+6.	
13.	Set HITL Detector Input 6 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 3 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x01 AND phaseStatusGroupPhaseOns.2 ≠ 0x10	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	

21.	WEND *Note:* Wait until controller reaches 1+13.	
22.	Set HITL Detector Input 3 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 4 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x01 AND phaseStatusGroupGreens.2 = 0x10 AND phaseStatusGroupVehCalls.1 AND 0x08 ≠ 0x08	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x01 AND phaseStatusGroupGreens.2 = 0x10 AND phaseStatusGroupVehCalls.1 AND 0x08 = 0x08 *Note*: Ensure that Phase 4 registers a Vehicle Call during 1+13 Green.	Pass/Fail
32.	Set HITL Detector Input 4 = Off	
33.	DELAY .2 Seconds	
Detector 4 calls Phase 4 when Phase 4 and Phase 12 are not Green (In 2+5 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 2 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x12 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 2+5.	
13.	Set HITL Detector Input 2 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 4 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x12 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x08 ≠ 0x08	
19.	DELAY 1 Second	

20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x12 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x08 = 0x08 *Note*: Ensure that Phase 4 registers a Vehicle Call during 2+5 Green.	Pass/Fail
23.	Set HITL Detector Input 4 = Off	
24.	DELAY .2 Seconds	
Detector 4 calls Phase 4 when Phase 4 and Phase 12 are not Green (In 3+5 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 4 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x14 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x08 ≠ 0x08	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x14 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x08 = 0x08 *Note*: Ensure that Phase 4 registers a Vehicle Call during 3+5 Green.	Pass/Fail
23.	Set HITL Detector Input 4 = Off	
24.	DELAY .2 Seconds	
Detector 4 calls Phase 4 when Phase 4 and Phase 12 are not Green (In 6+11 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	

5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 6 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x20 AND phaseStatusGroupPhaseOns.2 ≠ 0x04	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 6+11.	
22.	Set HITL Detector Input 6 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 4 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x04 AND phaseStatusGroupVehCalls.1 AND 0x08 ≠ 0x08	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x04 AND phaseStatusGroupVehCalls.1 AND 0x08 = 0x08 *Note*: Ensure that Phase 4 registers a Vehicle Call during 6+11 Green.	Pass/Fail
32.	Set HITL Detector Input 4 = Off	
33.	DELAY .2 Seconds	
Detector 4 calls Phase 4 when Phase 4 and Phase 12 are not Green (In 5+9 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	

5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 7 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x10 AND phaseStatusGroupPhaseOns.2 ≠ 0x01	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 5+9.	
22.	Set HITL Detector Input 7 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 4 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x10 AND phaseStatusGroupGreens.2 = 0x01 AND phaseStatusGroupVehCalls.1 AND 0x08 ≠ 0x08	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x10 AND phaseStatusGroupGreens.2 = 0x01 AND phaseStatusGroupVehCalls.1 AND 0x08 = 0x08 *Note*: Ensure that Phase 4 registers a Vehicle Call during 5+9 Green.	Pass/Fail
32.	Set HITL Detector Input 4 = Off	
33.	DELAY .2 Seconds	
Detector 4 extends Phase 4 during Phase 4		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	

5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 6 = On	
14.	DELAY .2 Seconds	
15.	GET ringStatus.1	
16.	WHILE ringStatus.1 AND 0x10 ≠ 0x10 (xxx1xxxx = maxout)	
17.	DELAY .2 Second	
18.	GET ringStatus.1	
19.	WEND *Note:* Wait for Max Out Indication. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
20.	GET phaseStatusGroupPhaseOns.1	
21.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x08 = 0x08 *Note:* Ensure that Max Out occurred on Phase 4.	Pass/Fail
22.	Set HITL Detector Input 4 = Off and 6 = Off	
23.	DELAY .2 Seconds	
Detector 4 extends Phase 12 during Phase 12		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 6 = On	

16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x20 AND phaseStatusGroupPhaseOns.2 ≠ 0x08	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 6+12.	
22.	Set HITL Detector Input 6 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 4 = On	
25.	DELAY .2 Seconds	
26.	Set HITL Detector Input 3 = On	
27.	DELAY .2 Seconds	
28.	GET ringStatus.1	
29.	WHILE ringStatus.1 AND 0x10 ≠ 0x10 (xxx1xxxx = maxout)	
30.	DELAY 1 Second	
31.	GET ringStatus.1	
32.	WEND *Note:* Wait for Max Out Indication. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
33.	GET phaseStatusGroupPhaseOns.2	
34.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.2 AND 0x04 = 0x04 *Note:* Ensure that Max Out occurred on Phase 12.	Pass/Fail
35.	Set HITL Detector Input 4 = Off and 3 = Off	
36.	DELAY .2 Seconds	
Teardown		
	PERFORM Detector Teardown – TC020 if not proceeding to another detector operation test case	
Test Case Results		
Tested By:	Date Tested	
Test Case Notes:		
Version History:	v1.00 04/14/06 Initial Draft – RDR v1.01 07/05/06 Updated notes – RDR v1.02 07/20/06 Implemented script and proofed – JJ	

Detector 5 Operations

Test Case: **TC005**	*Title*:	**Detector 5 Operations**
	Description:	Verifies the operation of Detector 5 to call Phase 2 under specific conditions and extend intervals 1625B, 1635, 1645B, 1735B, 1745B, 1835B, and 1845B.
	Constants:	
	Variables:	
	Pass/Fail Criteria:	The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.

Test Step Number	*Test Procedure*	*Results*
Setup		
	PERFORM Detector Operations Setup – TC019 if not already done so.	
Detector 5 No Call on Phase 2 when Overlap B is Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	GET ringStatus.1, ringStatus.2	
16.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
17.	DELAY 1 Second	
18.	GET ringStatus.1, ringStatus.2	
19.	WEND *Note:* Wait for 3+5 Green Rest.	
20.	GET overlapStatusGroupGreens.1 *Note:* overlapStatusGroupGreens is optional and a GET may return a noSuchName. This also assumes that Overlap B = 5+6+X.	
21.	IF RESPONSE ERROR = noError THEN	
22.	VERIFY that RESPONSE VALUE overlapStatusGroupGreens.1 AND 0x02 = 0x02 *Note:* Verifies that Overlap B = Green.	
23.	END IF	

24.	Set HITL Detector Input 5 = On	
25.	DELAY 3 Seconds	
26.	GET phaseStatusGroupVehCalls.1, phaseStatusGroupVehCalls.2	
27.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 AND 0x02 = 0x00. *Note:* Verifies that no call in entered on Phase 2.	Pass/Fail
28.	Set HITL Detector Input 5 = Off	
29.	DELAY .2 Seconds	
Detector 5 No Call on Phase 2 when Overlap B is not Green and a Call on Phase 3		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	GET overlapStatusGroupGreens.1 *Note:* overlapStatusGroupGreens is optional and a GET may return a noSuchName. This also assumes that Overlap B = 5+6+X.	
16.	IF RESPONSE ERROR = noError THEN	
17.	VERIFY that RESPONSE VALUE overlapStatusGroupGreens.1 AND 0x02 = 0x00 *Note:* Verifies that Overlap B = NOT Green	Pass/Fail
18.	END IF	
19.	Set HITL Detector Input 3 = On	
20.	DELAY 2 Seconds	
21.	Set HITL Detector Input 5 = On	
22.	DELAY 2 Seconds	
23.	GET phaseStatusGroupVehCalls.1	
24.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 AND 0x02 = 0x00 *Note:* Verifies that no call in entered on Phase 2.	Pass/Fail
25.	Set HITL Detector Input 5 = Off and 3 = Off	
26.	DELAY .2 Seconds	
Detector 5 No Call on Phase 2 when Overlap B is not Green and a Call on Phase 4		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	

3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	GET overlapStatusGroupGreens.1 *Note:* overlapStatusGroupGreens is optional and a GET may return a noSuchName. This also assumes that Overlap B = 5+6+X.	
16.	IF RESPONSE ERROR = noError THEN	
17.	VERIFY that RESPONSE VALUE overlapStatusGroupGreens.1 AND 0x02 = 0x00 *Note:* Verifies that Overlap B = NOT Green	
18.	END IF	
19.	Set HITL Detector Input 4 = On	
20.	DELAY 2 Seconds	
21.	Set HITL Detector Input 5 = On	
22.	DELAY 2 Seconds	
23.	GET phaseStatusGroupVehCalls.1	
24.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 AND 0x02 = 0x00 *Note:* Ensure that Phase 2 is not registering a Vehicle Call.	Pass/Fail
25.	Set HITL Detector Input 4 = Off and 5 = Off	
26.	DELAY .2 Seconds	
Detector 5 calls Phase 2 when Overlap B is not Green and no Calls on Phase 3 or Phase 4		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1,	

	phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	GET overlapStatusGroupGreens.1 *Note:* overlapStatusGroupGreens is optional and a GET may return a noSuchName. This also assumes that Overlap B = 5+6+X.	
16.	IF RESPONSE ERROR = noError THEN	
17.	VERIFY that RESPONSE VALUE overlapStatusGroupGreens.1 AND 0x02 = 0x00 *Note:* Verifies that Overlap B = NOT Green	
18.	END IF	
19.	Set HITL Detector Input 5 = On	
20.	DELAY .2 Seconds	
21.	GET phaseStatusGroupVehCalls.1	
22.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 AND 0x02 = 0x02 *Note:* Checks for call on Phase 2	Pass/Fail
23.	Set HITL Detector Input 5 = Off	
24.	DELAY .2 Seconds	
Detector 5 extends Phase 13 when Phase 13 is Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 4 = On	
16.	DELAY .2 Seconds	
17.	Set HITL Detector Input 5 = On	
18.	DELAY .2 Seconds	
19.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
20.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x01 AND phaseStatusGroupPhaseOns.2 ≠ 0x10	
21.	DELAY 1 Second	

22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WEND *Note:* Wait until controller reaches 1+13.	
24.	GET ringStatus.2	
25.	WHILE ringStatus.1 AND 0x10 ≠ 0x10 (xxx1xxxx = maxout)	
26.	DELAY 1 Second	
27.	GET ringStatus.2	
28.	WEND *Note:* Wait for Max Out Indication. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
29.	GET phaseStatusGroupPhaseOns.1	
30.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.2 AND 0x10 = 0x10 *Note:* Ensure that Max Out occurred on Phase 13.	Pass/Fail
31.	Set HITL Detector Input 5 = Off and 4 = Off	
32.	DELAY .2 Seconds	
Teardown		
	PERFORM Detector Teardown – TC020 if not proceeding to another detector operation test case	

Test Case Results				
Tested By:		Date Tested		Pass/Fail
Test Case Notes:	<notes>			
Version History:	v1.00 04/12/06 Initial Draft – RDR v1.01 07/05/06 Updated notes – RDR v1.02 07/20/06 Implemented script and proofed – JJ			

Detector 6 Operations

Test Case: **TC006**	*Title*: Description: Constants: Variables: Pass/Fail Criteria:	**Detector 6 Operations** Verifies the operation of Detector 6 to call and extend Phase 6. The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.
Test Step Number	*Test Procedure*	*Results*
Setup		
	PERFORM Detector Operations Setup – TC019 if not already done so.	
Detector 6 calls Phase 6 during 2+5 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	

5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 2 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x12 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 2+5.	
13.	Set HITL Detector Input 2 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 6 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x12 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x20 ≠ 0x20	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x12 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x20 = 0x20 *Note*: Ensure that Phase 6 registers a Vehicle Call during 2+5 Green.	Pass/Fail
23.	Set HITL Detector Input 6 = Off	
24.	DELAY .2 Seconds	
Detector 6 calls Phase 6 during 3+5 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	

14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 6 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x14 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x20 ≠ 0x20	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x14 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x20 = 0x20 *Note*: Ensure that Phase 6 registers a Vehicle Call during 3+5 Green.	Pass/Fail
23.	Set HITL Detector Input 6 = Off	
24.	DELAY .2 Seconds	
Detector 6 calls Phase 6 during 4+5 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 6 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x18 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x20 ≠ 0x20	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	

22.	VERIFY phaseStatusGroupGreens.1 = 0x18 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x20 = 0x20 *Note*: Ensure that Phase 6 registers a Vehicle Call during 4+5 Green.	Pass/Fail
23.	Set HITL Detector Input 6 = On	
24.	DELAY .2 Seconds	
Detector 6 calls Phase 6 during 5+9 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 7 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x10 AND phaseStatusGroupPhaseOns.2 ≠ 0x01	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 5+9.	
22.	Set HITL Detector Input 7 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 6 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x10 AND phaseStatusGroupGreens.2 = 0x01 AND phaseStatusGroupVehCalls.1 AND 0x20 ≠ 0x20	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	

31.	VERIFY phaseStatusGroupGreens.1 = 0x10 AND phaseStatusGroupGreens.2 = 0x01 AND phaseStatusGroupVehCalls.1 AND 0x20 = 0x20 *Note:* Ensure that Phase 6 registers a Vehicle Call during 5+9 Green.	Pass/Fail
32.	Set HITL Detector Input 6 = Off	
33.	DELAY .2 Seconds	
Detector 6 Calls Phase 6 during 1+7 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches or is in 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 6 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x41 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x20 ≠ 0x20	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x41 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x20 = 0x20 *Note*: Ensure that Phase 6 registers a Vehicle Call during 1+7 Green.	Pass/Fail
23.	Set HITL Detector Input 6 = Off	
24.	DELAY .2 Seconds	
Detector 6 calls Phase 6 during 1+8 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	

6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 6 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x81 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x20 ≠ 0x20	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x81 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x20 = 0x20 *Note:* Ensure that Phase 6 registers a Vehicle Call during 1+8 Green.	Pass/Fail
23.	Set HITL Detector Input 6 = Off	
24.	DELAY .2 Seconds	
Detector 6 calls Phase 6 during 2+16 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 2 = On	
16.	DELAY .2 Seconds	

17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x02 AND phaseStatusGroupPhaseOns.2 ≠ 0x80	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 2+16.	
22.	Set HITL Detector Input 2 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 6 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x80 AND phaseStatusGroupVehCalls.1 AND 0x20 ≠ 0x20	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x80 AND phaseStatusGroupVehCalls.1 AND 0x20 = 0x20 *Note:* Ensure that Phase 6 registers a Vehicle Call during 2+16 Green.	Pass/Fail
32.	Set HITL Detector Input 6 = Off	
33.	DELAY .2 Seconds	
Detector 6 calls Phase 6 during 2+15 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 2 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	

18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x02 AND phaseStatusGroupPhaseOns.2 ≠ 0x40	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 2+15.	
22.	Set HITL Detector Input 2 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 6 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x40 AND phaseStatusGroupVehCalls.1 AND 0x20 ≠ 0x20	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x40 AND phaseStatusGroupVehCalls.1 AND 0x20 = 0x20 *Note:* Ensure that Phase 6 registers a Vehicle Call during 2+15 Green.	Pass/Fail
32.	Set HITL Detector Input 6 = Off	
33.	DELAY .2 Seconds	
Detector 6 calls Phase 6 during 1+13 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 6 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+6.	
13.	Set HITL Detector Input 6 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 2 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	

18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x01 AND phaseStatusGroupPhaseOns.2 ≠ 0x10	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 1+13.	
22.	Set HITL Detector Input 2 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 6 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x01 AND phaseStatusGroupGreens.2 = 0x10 AND phaseStatusGroupVehCalls.1 AND 0x20 ≠ 0x20	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x01 AND phaseStatusGroupGreens.2 = 0x10 AND phaseStatusGroupVehCalls.1 AND 0x20 = 0x20 *Note:* Ensure that Phase 6 registers a Vehicle Call during 1+13 Green.	Pass/Fail
32.	Set HITL Detector Input 6 = Off	
33.	DELAY .2 Seconds	
Detector 6 extends Phase 6 during 1+6		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 6 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+6.	
13.	Set HITL Detector Input 7 = On	
14.	DELAY .2 Seconds	
15.	GET ringStatus.2	
16.	WHILE ringStatus.2 AND 0x10 ≠ 0x10 (xxx1xxxx = maxout)	
17.	DELAY 1 Second	
18.	GET ringStatus.2	

19.	WEND *Note:* Wait for Max Out Indication. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13			
20.	GET phaseStatusGroupPhaseOns.1			
21.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x20 = 0x20 *Note:* Ensure that Max Out occurred on Phase 6.			Pass/Fail
22.	Set HITL Detector Input 6 = Off and 7 = Off			
23.	DELAY .2 Seconds			
Teardown				
	PERFORM Detector Teardown – TC020 if not proceeding to another detector operation test case			
Test Case Results				
Tested By:		Date Tested		
Test Case Notes:				
Version History:	v1.00 05/05/06 Initial Draft – RDR v1.01 07/05/06 Added 5+9, 2+16, 2+15, and 1+13. Updated notes – RDR v1.02 07/20/06 Implemented script and proofed – JJ			

Detector 7 Operations

Test Case: **TC007**	*Title*:	**Detector 7 Operations**	
	Description:	Verifies the operation of Detector 7 to call and extend Phase 7 under specific conditions and extend interval 1725B.	
	Constants:		
	Variables:		
	Pass/Fail Criteria:	The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*		*Results*
Setup			
	PERFORM Detector Operations Setup – TC019 if not already done so.		
Detector 7 calls Phase 7 when Phase 7 and Phase 15 are not Green (In 1+6 Green)			
1.	GET ringStatus.1, ringStatus.2		
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03		
3.	DELAY 1 Second		
4.	GET ringStatus.1, ringStatus.2		
5.	WEND *Note:* Loop until controller rests in green somewhere.		
6.	Set HITL Detector Input 6 = On		
7.	DELAY .2 Seconds		
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2		
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0x00		
10.	DELAY 1 Second		
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2		

12.	WEND *Note:* Wait until controller reaches 1+6.	
13.	Set HITL Detector Input 6 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 7 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x21 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x40 ≠ 0x40	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x21 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x40 = 0x40 *Note*: Ensure that Phase 7 registers a Vehicle Call during 1+6 Green.	Pass/Fail
23.	Set HITL Detector Input 7 = Off	
24.	DELAY .2 Seconds	
Detector 7 calls Phase 7 when Phase 7 and Phase 15 are not Green (In 1+8 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = On	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 7 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x81 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x40 ≠ 0x40	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	

21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x81 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x40 = 0x40 *Note*: Ensure that Phase 7 registers a Vehicle Call during 1+8 Green.	Pass/Fail
23.	Set HITL Detector Input 7 = Off	
24.	DELAY .2 Seconds	
Detector 7 calls Phase 7 when Phase 7 and Phase 15 are not Green (In 2+16 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 2 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x02 AND phaseStatusGroupPhaseOns.2 ≠ 0x80	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 2+16.	
22.	Set HITL Detector Input 2 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 7 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x80 AND phaseStatusGroupVehCalls.1 AND 0x40 ≠ 0x40	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x02 AND	Pass/Fail

	phaseStatusGroupGreens.2 = 0x80 AND phaseStatusGroupVehCalls.1 AND 0x40 = 0x40 *Note*: Ensure that Phase 7 registers a Vehicle Call during 2+16 Green.	
32.	Set HITL Detector Input 7 = Off	
33.	DELAY .2 Seconds	
Detector 7 calls Phase 7 when Phase 7 and Phase 15 are not Green (In 1+13 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 6 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+6.	
13.	Set HITL Detector Input 6 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 3 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x01 AND phaseStatusGroupPhaseOns.2 ≠ 0x10	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 1+13.	
22.	Set HITL Detector Input 3 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 7 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x01 AND phaseStatusGroupGreens.2 = 0x10 AND phaseStatusGroupVehCalls.1 AND 0x40 ≠ 0x40	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x01 AND phaseStatusGroupGreens.2 = 0x10 AND	Pass/Fail

	phaseStatusGroupVehCalls.1 AND 0x40 = 0x40 *Note*: Ensure that Phase 7 registers a Vehicle Call during 1+13 Green.	
32.	Set HITL Detector Input 7 = Off	
33.	DELAY .2 Seconds	
Detector 7 calls Phase 7 when Phase 7 and Phase 15 are not Green (In 2+5 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 2 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x12 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 2+5.	
13.	Set HITL Detector Input 2 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 7 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x12 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x40 ≠ 0x40	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x12 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x40 = 0x40 *Note*: Ensure that Phase 7 registers a Vehicle Call during 2+5 Green.	Pass/Fail
23.	Set HITL Detector Input 7 = Off	
24.	DELAY .2 Seconds	
Detector 7 calls Phase 7 when Phase 7 and Phase 15 are not Green (In 3+5 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	

7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 7 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x14 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x40 ≠ 0x40	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x14 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x40 = 0x40 *Note*: Ensure that Phase 7 registers a Vehicle Call during 3+5 Green.	Pass/Fail
23.	Set HITL Detector Input 7 = Off	
24.	DELAY .2 Seconds	
Detector 7 calls Phase 7 when Phase 7 and Phase 15 are not Green (In 4+5 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 7 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2,	

	phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x18 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x40 ≠ 0x40	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x18 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x40 = 0x40 *Note*: Ensure that Phase 7 registers a Vehicle Call during 4+5 Green.	Pass/Fail
23.	Set HITL Detector Input 7 = Off	
24.	DELAY .2 Seconds	
Detector 7 calls Phase 7 when Phase 7 and Phase 15 are not Green (In 6+12 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 6 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x20 AND phaseStatusGroupPhaseOns.2 ≠ 0x08	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 6+12.	
22.	Set HITL Detector Input 6 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 7 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x20 AND	

	phaseStatusGroupGreens.2 = 0x08 AND phaseStatusGroupVehCalls.1 AND 0x40 ≠ 0x40	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x08 AND phaseStatusGroupVehCalls.1 AND 0x40 = 0x40 *Note*: Ensure that Phase 7 registers a Vehicle Call during 6+12 Green.	Pass/Fail
32.	Set HITL Detector Input 7 = Off	
33.	DELAY .2 Seconds	
Detector 7 calls Phase 7 when Phase 7 and Phase 15 are not Green (In 6+11 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 6 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x20 AND phaseStatusGroupPhaseOns.2 ≠ 0x04	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 6+11.	
22.	Set HITL Detector Input 6 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 7 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x04 AND	

	phaseStatusGroupVehCalls.1 AND 0x40 ≠ 0x40	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x04 AND phaseStatusGroupVehCalls.1 AND 0x40 = 0x40 *Note*: Ensure that Phase 7 registers a Vehicle Call during 6+11 Green.	Pass/Fail
32.	Set HITL Detector Input 7 = Off	
33.	DELAY .2 Seconds	
Detector 7 calls Phase 7 when Phase 7 and Phase 15 are not Green (In 5+9 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 8 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x10 AND phaseStatusGroupPhaseOns.2 ≠ 0x01	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 5+9.	
22.	Set HITL Detector Input 8 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 7 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x10 AND phaseStatusGroupGreens.2 = 0x01 AND phaseStatusGroupVehCalls.1 AND 0x40 ≠ 0x40	

28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x10 AND phaseStatusGroupGreens.2 = 0x01 AND phaseStatusGroupVehCalls.1 AND 0x40 = 0x40 *Note*: Ensure that Phase 7 registers a Vehicle Call during 5+9 Green.	Pass/Fail
32.	Set HITL Detector Input 7 = Off	
33.	DELAY .2 Seconds	
Detector 7 extends Phase 7 during Phase 7		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 8 = On	
14.	DELAY .2 Seconds	
15.	GET ringStatus.2	
16.	WHILE ringStatus.2 AND 0x10 ≠ 0x10 (xxx1xxxx = maxout)	
17.	DELAY .2 Second	
18.	GET ringStatus.2	
19.	WEND *Note:* Wait for Max Out Indication. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
20.	GET phaseStatusGroupPhaseOns.1	
21.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x40 = 0x40 *Note:* Ensure that Max Out occurred on Phase 7.	Pass/Fail
22.	Set HITL Detector Input 7 = Off and 8 = Off	
23.	DELAY .2 Seconds	
Detector 7 extends Phase 15 during Phase 15		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND	

	Note: Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 2 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x02 AND phaseStatusGroupPhaseOns.2 ≠ 0x40	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 2+15.	
22.	Set HITL Detector Input 2 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 7 = On	
25.	DELAY .2 Seconds	
26.	Set HITL Detector Input 3 = On	
27.	DELAY .2 Seconds	
28.	GET ringStatus.2	
29.	WHILE ringStatus.2 AND 0x10 ≠ 0x10 (xxx1xxxx = maxout)	
30.	DELAY 1 Second	
31.	GET ringStatus.2	
32.	WEND *Note:* Wait for Max Out Indication. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
33.	GET phaseStatusGroupPhaseOns.2	
34.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.2 AND 0x40 = 0x40 *Note:* Ensure that Max Out occurred on Phase 15.	Pass/Fail
35.	Set HITL Detector Input 7 = Off and 3 = Off	
36.	DELAY .2 Seconds	

Teardown		
	PERFORM Detector Teardown – TC020 if not proceeding to another detector operation test case	
Test Case Results		
Tested By:		Date Tested		
Test Case Notes:		
Version History:	v1.00 05/05/06 Initial Draft – RDR v1.01 07/05/06 Updated notes – RDR v1.02 07/20/06 Implemented script and proofed – JJ	

Detector 8 Operations

Test Case: **TC008**	*Title*: Description: Constants: Variables: Pass/Fail Criteria:	**Detector 8 Operations** Verifies the operation of Detector 8 to call and extend Phase 8 under specific conditions and extend interval 1825B. The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.
Test Step Number	*Test Procedure*	*Results*
Setup		
	PERFORM Detector Operations Setup – TC019 if not already done so.	
Detector 8 calls Phase 8 when Phase 8 and Phase 16 are not Green (In 1+6 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 6 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+6.	
13.	Set HITL Detector Input 6 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 8 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x21 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x80 ≠ 0x80	

19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x21 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x80 = 0x80 *Note*: Ensure that Phase 8 registers a Vehicle Call during 1+6 Green.	Pass/Fail
23.	Set HITL Detector Input 8 = Off	
24.	DELAY .2 Seconds	
Detector 8 calls Phase 8 when Phase 8 and Phase 16 are not Green (In 1+7 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = On	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 8 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x41 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x80 ≠ 0x80	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x41 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x80 = 0x80 *Note*: Ensure that Phase 8 registers a Vehicle Call during 1+7 Green.	Pass/Fail
23.	Set HITL Detector Input 8 = Off	
24.	DELAY .2 Seconds	
Detector 8 calls Phase 8 when Phase 8 and Phase 16 are not Green (In 2+15 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	

4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 2 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x02 AND phaseStatusGroupPhaseOns.2 ≠ 0x40	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 2+15.	
22.	Set HITL Detector Input 2 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 8 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x40 AND phaseStatusGroupVehCalls.1 AND 0x80 ≠ 0x80	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x40 AND phaseStatusGroupVehCalls.1 AND 0x80 = 0x80 *Note*: Ensure that Phase 8 registers a Vehicle Call during 2+15 Green.	Pass/Fail
32.	Set HITL Detector Input 8 = Off	
33.	DELAY .2 Seconds	
Detector 8 calls Phase 8 when Phase 8 and Phase 16 are not Green (In 1+13 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	

5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 6 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+6.	
13.	Set HITL Detector Input 6 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 2 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x01 AND phaseStatusGroupPhaseOns.2 ≠ 0x10	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 1+13.	
22.	Set HITL Detector Input 2 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 8 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x01 AND phaseStatusGroupGreens.2 = 0x10 AND phaseStatusGroupVehCalls.1 AND 0x80 ≠ 0x80	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x01 AND phaseStatusGroupGreens.2 = 0x10 AND phaseStatusGroupVehCalls.1 AND 0x80 = 0x80 *Note*: Ensure that Phase 8 registers a Vehicle Call during 1+13 Green.	Pass/Fail
32.	Set HITL Detector Input 8 = Off	
33.	DELAY .2 Seconds	
Detector 8 calls Phase 8 when Phase 8 and Phase 16 are not Green (In 2+5 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND	

	Note: Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 2 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x12 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 2+5.	
13.	Set HITL Detector Input 2 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 8 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x12 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x80 ≠ 0x80	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x12 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x80 = 0x80 *Note*: Ensure that Phase 8 registers a Vehicle Call during 2+5 Green.	Pass/Fail
23.	Set HITL Detector Input 8 = Off	
24.	DELAY .2 Seconds	
Detector 8 calls Phase 8 when Phase 8 and Phase 16 are not Green (In 3+5 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	

15.	Set HITL Detector Input 8 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x14 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x80 ≠ 0x80	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x14 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x80 = 0x80 *Note*: Ensure that Phase 8 registers a Vehicle Call during 3+5 Green.	Pass/Fail
23.	Set HITL Detector Input 8 = Off	
24.	DELAY .2 Seconds	
Detector 8 calls Phase 8 when Phase 8 and Phase 16 are not Green (In 4+5 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 8 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x18 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x80 ≠ 0x80	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x18 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x80 = 0x80	Pass/Fail

	Note: Ensure that Phase 8 registers a Vehicle Call during 4+5 Green.	
23.	Set HITL Detector Input 8 = Off	
24.	DELAY .2 Seconds	
Detector 8 calls Phase 8 when Phase 8 and Phase 16 are not Green (In 6+12 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 6 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x20 AND phaseStatusGroupPhaseOns.2 ≠ 0x08	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 6+12.	
22.	Set HITL Detector Input 6 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 8 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x08 AND phaseStatusGroupVehCalls.1 AND 0x80 ≠ 0x80	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x08 AND phaseStatusGroupVehCalls.1 AND 0x80 = 0x80 *Note*: Ensure that Phase 8 registers a Vehicle Call during 6+12 Green.	Pass/Fail

32.	Set HITL Detector Input 8 = Off	
33.	DELAY .2 Seconds	
Detector 8 calls Phase 8 when Phase 8 and Phase 16 are not Green (In 6+11 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 6 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x20 AND phaseStatusGroupPhaseOns.2 ≠ 0x04	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 6+11.	
22.	Set HITL Detector Input 6 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 8 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x04 AND phaseStatusGroupVehCalls.1 AND 0x80 ≠ 0x80	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x04 AND phaseStatusGroupVehCalls.1 AND 0x80 = 0x80 *Note*: Ensure that Phase 8 registers a Vehicle Call during 6+11 Green.	Pass/Fail
32.	Set HITL Detector Input 8 = Off	

33.	DELAY .2 Seconds	
Detector 8 calls Phase 8 when Phase 8 and Phase 16 are not Green (In 5+9 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 2 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x12 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 2+5.	
13.	Set HITL Detector Input 2 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 6 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x10 AND phaseStatusGroupPhaseOns.2 ≠ 0x01	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 5+9.	
22.	Set HITL Detector Input 6 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 8 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x10 AND phaseStatusGroupGreens.2 = 0x01 AND phaseStatusGroupVehCalls.1 AND 0x80 ≠ 0x80	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x10 AND phaseStatusGroupGreens.2 = 0x01 AND phaseStatusGroupVehCalls.1 AND 0x80 = 0x80 *Note*: Ensure that Phase 8 registers a Vehicle Call during 5+9 Green.	Pass/Fail
32.	Set HITL Detector Input 8 = Off	
33.	DELAY .2 Seconds	

Detector 8 extends Phase 8 during Phase 8		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 7 = On	
14.	DELAY .2 Seconds	
15.	GET ringStatus.2	
16.	WHILE ringStatus.2 AND 0x10 ≠ 0x10 (xxx1xxxx = maxout)	
17.	DELAY .2 Second	
18.	GET ringStatus.2	
19.	WEND *Note:* Wait for Max Out Indication. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
20.	GET phaseStatusGroupPhaseOns.1	
21.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x40 = 0x40 *Note:* Ensure that Max Out occurred on Phase 8.	Pass/Fail
22.	Set HITL Detector Input 8 = Off and 7 = Off	
23.	DELAY .2 Seconds	
Detector 8 extends Phase 16 during Phase 16		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	

12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 2 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x02 AND phaseStatusGroupPhaseOns.2 ≠ 0x80	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 2+16.	
22.	Set HITL Detector Input 2 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 8 = On	
25.	DELAY .2 Seconds	
26.	Set HITL Detector Input 3 = On	
27.	DELAY .2 Seconds	
28.	GET ringStatus.2	
29.	WHILE ringStatus.2 AND 0x10 ≠ 0x10 (xxx1xxxx = maxout)	
30.	DELAY 1 Second	
31.	GET ringStatus.2	
32.	WEND *Note:* Wait for Max Out Indication. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
33.	GET phaseStatusGroupPhaseOns.2	
34.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.2 AND 0x40 = 0x40 *Note:* Ensure that Max Out occurred on Phase 16.	Pass/Fail
35.	Set HITL Detector Input 8 = Off and 3 = Off	
36.	DELAY .2 Seconds	
Teardown		
	PERFORM Detector Teardown – TC020 if not proceeding to another detector operation test case	
	Test Case Results	
Tested By:	Date Tested	
Test Case Notes:		
Version History:	v1.00 05/08/06 Initial Draft – RDR v1.01 07/05/06 Updated notes – RDR v1.02 07/21/06 Implemented script and proofed – JJ	

Detector 9 Operations

Test Case: **TC009**	Title:	**Detector 9 Operations**	
	Description:	Verifies the operation of Detector 9 to call Phase 6 under specific conditions, extend Phase 2 under specific conditions, and extend intervals 2516B, 2517B, 2518B, 3517B, 3518B, 4517B, and 4518B.	
	Constants:		
	Variables:		
	Pass/Fail Criteria:	The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	

Test Step Number	*Test Procedure*	*Results*
Setup		
	PERFORM Detector Operations Setup – TC019 if not already done so.	
Detector 9 No Call on Phase 6 when Overlap A is Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 6 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+6.	
13.	Set HITL Detector Input 6 = Off	
14.	DELAY .2 Seconds	
15.	GET ringStatus.1, ringStatus.2	
16.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
17.	DELAY 1 Second	
18.	GET ringStatus.1, ringStatus.2	
19.	WEND *Note:* Wait for 1+6 Green Rest.	
20.	GET overlapStatusGroupGreens.1 *Note:* overlapStatusGroupGreens is optional and a GET may return a noSuchName. This also assumes that Overlap A = 1+2+X.	Pass/Fail
21.	IF RESPONSE ERROR = noError THEN	
22.	VERIFY that RESPONSE VALUE overlapStatusGroupGreens.1 AND 0x01 = 0x01 *Note:* Verifies that Overlap A = Green.	Pass/Fail

23.	ENDIF	
24.	Set HITL Detector Input 9 = On	
25.	DELAY 2 Seconds (2 full seconds)	
26.	GET phaseStatusGroupVehCalls.1, phaseStatusGroupVehCalls.2	
27.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 = 0x00 AND phaseStatusGroupVehCalls.2 = 0x00 *Note:* Verify that controller does not register call.	Pass/Fail
28.	Set HITL Detector Input 9 = Off	
29.	DELAY .2 Seconds	
Detector 9 No Call on Phase 6 when Overlap A is not Green and a Call on Phase 7		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	GET overlapStatusGroupGreens.1 *Note:* overlapStatusGroupGreens is optional and a GET may return a noSuchName. This also assumes that Overlap A = 1+2+X.	Pass/Fail
16.	IF RESPONSE ERROR = noError THEN	
17.	VERIFY that RESPONSE VALUE overlapStatusGroupGreens.1 AND 0x01 = 0x00 *Note:* Verifies that Overlap A = NOT Green	Pass/Fail
18.	ENDIF	
19.	Set HITL Detector Input 7 = On	
20.	DELAY .2 Seconds	
21.	Set HITL Detector Input and 9 = On	
22.	DELAY 2 Seconds (2 full seconds)	
23.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 AND 0x20 = 0 *Note:* Checks for no call on Phase 6	Pass/Fail
24.		
25.	Set HITL Detector Input 9 = Off and 7 = Off	
26.	DELAY .2 Seconds	
Detector 9 No Call on Phase 6 when Overlap A is not Green and a Call on Phase 8		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	

3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	GET overlapStatusGroupGreens.1 *Note:* overlapStatusGroupGreens is optional and a GET may return a noSuchName. This also assumes that Overlap A = 1+2+X.	Pass/Fail
16.	IF RESPONSE ERROR = noError THEN	
17.	VERIFY that RESPONSE VALUE overlapStatusGroupGreens.1 AND 0x01 = 0x00 *Note:* Verifies that Overlap A = NOT Green	Pass/Fail
18.	ENDIF	
19.	Set HITL Detector Input 8 = On	
20.	DELAY .2 Seconds	
21.	Set HITL Detector Input 9 = On	
22.	DELAY 2 Seconds (2 full seconds)	
23.	GET phaseStatusGroupVehCalls.1	
24.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 AND 0x20 = 0 *Note:* Checks for no call on Phase 6	Pass/Fail
25.	Set HITL Detector Input 9 = Off and 8 = Off	
26.	DELAY .2 Seconds	
Detector 9 Calls Phase 6 when Overlap A is not Green and no Calls on Phase 7 or Phase 8		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	

12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	GET overlapStatusGroupGreens.1 *Note:* overlapStatusGroupGreens is optional and a GET may return a noSuchName. This also assumes that Overlap A = 1+2+X.	Pass/Fail
16.	IF RESPONSE ERROR = noError THEN	
17.	VERIFY that RESPONSE VALUE overlapStatusGroupGreens.1 AND 0x01 = 0x00 *Note:* Verifies that Overlap A = NOT Green.	Pass/Fail
18.	ENDIF	
19.	Set HITL Detector Input 9 = On	
20.	DELAY 2 Seconds (2 full seconds)	
21.	GET phaseStatusGroupVehCalls.1	
22.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 AND 0x20 = 0x20 *Note:* Checks for call on Phase 6	Pass/Fail
23.	Set HITL Detector Input 9 = Off	
24.	DELAY .2 Seconds	
Detector 9 does not Extends Phase 2 when Phase 2 is Green and there are no Calls on Phase 3 or Phase 4		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 2 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x12 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 2+5.	
13.	Set HITL Detector Input 2 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 9 = On *Note:* To possible extend	
16.	DELAY .2 Seconds	
17.	Set HITL Detector Input 2 = On *Note:* To force extensions	
18.	DELAY .2 Seconds	

19.	Set HITL Detector Input 8 = On *Note:* When opposing call exists	
20.	DELAY .2 Seconds	
21.	Set HITL Detector Input 2 = Off *Note:* To check whether possible extensions are due to Detector 9.	
22.	DELAY .2 Seconds	
23.	GET ringStatus.1	
24.	WHILE ringStatus.1 AND 0x08 ≠ 0x08 (xxxx1xxx = gap out)	
25.	DELAY 1 Second	
26.	GET ringStatus.1, ringStatus.2	
27.	WEND *Note:* Wait for Gap Out indication on Ring 1. Ring 1 = ***2***, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
28.	GET phaseStatusGroupPhaseOns.1	
29.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x02 = 0x02 *Note:* Verify that Gap Out indication occurred on Phase 2.	Pass/Fail
30.	Set HITL Detector Input 8 and 9 = Off	
31.	DELAY .2 Seconds	
Detector 9 extends Phase 2 when Phase 2 is Green and a Call on Phase 3		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 2 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x12 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 2+5.	
13.	Set HITL Detector Input 2 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 9 = On *Note:* To possible extend	
16.	DELAY .2 Seconds	
17.	Set HITL Detector Input 2 = On *Note:* To force extensions	
18.	DELAY .2 Seconds	

19.	Set HITL Detector Input 3 = On *Note:* When opposing call exists	
20.	DELAY .2 Seconds	
21.	Set HITL Detector Input 2 = Off *Note:* To check whether possible extensions are due to Detector 9.	
22.	DELAY .2 Seconds	
23.	GET ringStatus.1	
24.	WHILE ringStatus.1 AND 0x10 ≠ 0x10 (xxx1xxxx = maxout)	
25.	DELAY 1 Second	
26.	GET ringStatus.1	
27.	WEND *Note:* Wait for Max Out Indication on Ring 1. Ring 1 = ***2***, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
28.	GET phaseStatusGroupPhaseOns.1	
29.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x02 = 0x02 *Note:* Verify that Max Out indication occurred on Phase 2	Pass/Fail
30.	Set HITL Detector Input 3 and 9 = Off	
31.	DELAY .2 Seconds	
Detector 9 extends Phase 2 when Phase 2 is Green and a Call on Phase 4		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 2 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x12 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 2+5.	
13.	Set HITL Detector Input 2 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 9 = On *Note:* To possible extend	
16.	DELAY .2 Seconds	
17.	Set HITL Detector Input 2 = On *Note:* To force extensions	
18.	DELAY .2 Seconds	

19.	Set HITL Detector Input 4 = On *Note:* When opposing call exists	
20.	DELAY .2 Seconds	
21.	Set HITL Detector Input 2 = Off *Note:* To check whether possible extensions are due to Detector 9.	
22.	DELAY .2 Seconds	
23.	GET ringStatus.1	
24.	WHILE ringStatus.1 AND 0x10 ≠ 0x10 (xxx1xxxx = maxout)	
25.	DELAY 1 Second	
26.	GET ringStatus.1	
27.	WEND *Note:* Wait for Max Out Indication on Ring 1 Ring 1 = ***2***, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
28.	GET phaseStatusGroupPhaseOns.1	
29.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x02 = 0x02 *Note:* Verify that Max Out indication occurred on Phase 2	Pass/Fail
30.	Set HITL Detector Input 4 and 9 = Off	
31.	DELAY .2 Seconds	
Detector 9 extends Phase 9 when Phase 9 is Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 7 = On	
16.	DELAY .2 Seconds	
17.	Set HITL Detector Input 9 = On	
18.	DELAY .2 Seconds	
19.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
20.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x10 AND phaseStatusGroupPhaseOns.2 ≠ 0X01	
21.	DELAY 1 Second	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	

23.	WEND *Note:* Wait until controller reaches 5+9.	
24.	Set HITL Detector Input 7 = Off	
25.	DELAY .2 Seconds	
26.	GET ringStatus.1	
27.	WHILE ringStatus.1 AND 0x10 ≠ 0x10 (xxx1xxxx = maxout)	
28.	DELAY 1 Second	
29.	GET ringStatus.1	
30.	WEND *Note:* Wait for Max Out Indication. Ring 1 = 2, 3, 4, ***9***, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
31.	GET phaseStatusGroupPhaseOns.2	
32.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.2 AND 0x01 = 0x01 *Note:* Ensure that Max Out occurred on Phase 9.	Pass/Fail
33.	Set HITL Detector Input 9 = Off	
34.	DELAY .2 Seconds	
Teardown		
	PERFORM Detector Teardown – TC020 if not proceeding to another detector operation test case	

		Test Case Results		
Tested By:		Date Tested		Pass/Fail
Test Case Notes:	<notes>			
Version History:	v1.00 05/09/06 Initial Draft – RDR v1.01 07/05/06 Updated notes – RDR v1.02 07/17/06 Implemented script and proofed – JJ			

Detector 10 Operations

Test Case: **TC010**	*Title*: **Detector 10 Operations** Description: Verifies the operation of Detector 10 to call Phase 6 under specific conditions, extend Phase 2 under specific conditions, and extend intervals 2516B, 2517B, 2518B, 3517B, 3518B, 4517B, and 4518B. Constants: Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
Setup		
	PERFORM Detector Operations Setup – TC019 if not already done so.	
Detector 10 No Call on Phase 6 when Overlap A is Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	

4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 6 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+6.	
13.	Set HITL Detector Input 6 = Off	
14.	DELAY .2 Seconds	
15.	GET ringStatus.1, ringStatus.2	
16.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
17.	DELAY 1 Second	
18.	GET ringStatus.1, ringStatus.2	
19.	WEND *Note:* Wait for 1+6 Green Rest.	
20.	GET overlapStatusGroupGreens.1 *Note:* overlapStatusGroupGreens is optional and a GET may return a noSuchName. This also assumes that Overlap A = 1+2+X.	
21.	IF RESPONSE ERROR = noError THEN	
22.	VERIFY that RESPONSE VALUE overlapStatusGroupGreens.1 AND 0x01 = 0x01 *Note:* Verifies that Overlap A = Green	Pass/Fail
23.	END IF	
24.	Set HITL Detector Input 10 = On	
25.	DELAY 2 Seconds	
26.	GET phaseStatusGroupVehCalls.1, phaseStatusGroupVehCalls.2	
27.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 = 0x00 AND phaseStatusGroupVehCalls.2 = 0x00 *Note:* Verify that controller does not register call.	Pass/Fail
28.	Set HITL Detector Input 10 = Off	
29.	DELAY .2 Seconds	
Detector 10 No Call on Phase 6 when Overlap A is not Green and a Call on Phase 7		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	

9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	GET overlapStatusGroupGreens.1 *Note:* overlapStatusGroupGreens is optional and a GET may return a noSuchName. This also assumes that Overlap A = 1+2+X	
16.	IF RESPONSE ERROR = noError THEN	
17.	VERIFY that RESPONSE VALUE overlapStatusGroupGreens.1 AND 0x01 = 0x00 *Note:* Verifies that Overlap A = NOT Green.	Pass/Fail
18.	END IF	
19.	Set HITL Detector Input 7 = On	
20.	DELAY .2 Seconds	
21.	Set HITL Detector Input 10 = On	
22.	DELAY 2 Seconds	
23.	GET phaseStatusGroupVehCalls.1	
24.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 AND 0x20 = 0 *Note:* Checks for no call on Phase 6	Pass/Fail
25.	Set HITL Detector Input 10 = Off and 7 = Off	
26.	DELAY .2 Seconds	
Detector 10 No Call on Phase 6 when Overlap A is not Green and a Call on Phase 8		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	

15.	GET overlapStatusGroupGreens.1 *Note:* overlapStatusGroupGreens is optional and a GET may return a noSuchName. This also assumes that Overlap A = 1+2+X	
16.	IF RESPONSE ERROR = noError THEN	
17.	VERIFY that RESPONSE VALUE overlapStatusGroupGreens.1 AND 0x01 = 0x00 *Note:* Verifies that Overlap A = NOT Green.	Pass/Fail
18.	END IF	
19.	Set HITL Detector Input 8 = On	
20.	DELAY .2 Seconds	
21.	Set HITL Detector Input 10 = On	
22.	DELAY 2 Seconds	
23.	GET phaseStatusGroupVehCalls.1	
24.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 AND 0x20 = 0 *Note:* Checks for no call on Phase 6	Pass/Fail
25.	Set HITL Detector Input 10 = Off and 8 = Off	
26.	DELAY .2 Seconds	
Detector 10 Calls Phase 6 when Overlap A is not Green and no Calls on Phase 7 or Phase 8		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	GET overlapStatusGroupGreens.1 *Note:* overlapStatusGroupGreens is optional and a GET may return a noSuchName. This also assumes that Overlap A = 1+2+X	
16.	IF RESPONSE ERROR = noError THEN	
17.	VERIFY that RESPONSE VALUE overlapStatusGroupGreens.1 AND 0x01 = 0x00 *Note:* Verifies that Overlap A = NOT Green.	Pass/Fail
18.	END IF	
19.	Set HITL Detector Input 10 = On	
20.	DELAY 2 Seconds	

21.	GET phaseStatusGroupVehCalls.1	
22.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 AND 0x20 = 0x20 *Note:* Checks for call on Phase 6	Pass/Fail
23.	Set HITL Detector Input 10 = Off	
24.	DELAY .2 Seconds	
Detector 10 does not Extends Phase 2 when Phase 2 is Green and there are no Calls on Phase 3 or Phase 4		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 2 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x12 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 2+5.	
13.	Set HITL Detector Input 2 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 10 = On *Note:* To possible extend	
16.	DELAY .2 Seconds	
17.	Set HITL Detector Input 2 = On *Note:* To force extensions	
18.	DELAY .2 Seconds	
19.	Set HITL Detector Input 8 = On *Note:* When opposing call exists	
20.	DELAY .2 Seconds	
21.	Set HITL Detector Input 2 = Off *Note:* To check whether possible extensions are due to Detector 10.	
22.	DELAY .2 Seconds	
23.	GET ringStatus.1	
24.	WHILE ringStatus.1 AND 0x08 ≠ 0x08 (xxxx1xxx = gap out)	
25.	DELAY 1 Second	
26.	GET ringStatus.1, ringStatus.2	
27.	WEND *Note:* Wait for Gap Out indication on Ring 1 Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	

28.	GET phaseStatusGroupPhaseOns.1	
29.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x02 = 0x02 *Note:* Verify that Gap Out indication occurred on Phase 2	Pass/Fail
30.	Set HITL Detector Input 8 and 9 = Off	
31.	DELAY .2 Seconds	
Detector 10 extends Phase 2 when Phase 2 is Green and a Call on Phase 3		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 2 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x12 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 2+5.	
13.	Set HITL Detector Input 2 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 10 = On *Note:* To possible extend	
16.	DELAY .2 Seconds	
17.	Set HITL Detector Input 2 = On *Note:* To force extensions	
18.	DELAY .2 Seconds	
19.	Set HITL Detector Input 3 = On *Note:* When opposing call exists	
20.	DELAY .2 Seconds	
21.	Set HITL Detector Input 2 = Off *Note:* To check whether possible extensions are due to Detector 10.	
22.	DELAY .2 Seconds	
23.	GET ringStatus.1	
24.	WHILE ringStatus.1 AND 0x10 ≠ 0x10 (xxx1xxxx = maxout)	
25.	DELAY 1 Second	
26.	GET ringStatus.1	
27.	WEND *Note:* Wait for Max Out Indication on Ring 1 Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
28.	GET phaseStatusGroupPhaseOns.1	

29.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x02 = 0x02 *Note:* Verify that Max Out indication occurred on Phase 2	Pass/Fail
30.	Set HITL Detector Input 3 and 9 = Off	
31.	DELAY .2 Seconds	
Detector 10 extends Phase 2 when Phase 2 is Green and a Call on Phase 4		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 2 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x12 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 2+5.	
13.	Set HITL Detector Input 2 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 10 = On *Note:* To possible extend	
16.	DELAY .2 Seconds	
17.	Set HITL Detector Input 2 = On *Note:* To force extensions	
18.	DELAY .2 Seconds	
19.	Set HITL Detector Input 4 = On *Note:* When opposing call exists	
20.	DELAY .2 Seconds	
21.	Set HITL Detector Input 2 = Off *Note:* To check whether possible extensions are due to Detector 10	
22.	DELAY .2 Seconds	
23.	GET ringStatus.1	
24.	WHILE ringStatus.1 AND 0x10 ≠ 0x10 (xxx1xxxx = maxout)	
25.	DELAY 1 Second	
26.	GET ringStatus.1	
27.	WEND *Note:* Wait for Max Out Indication on Ring 1 Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
28.	GET phaseStatusGroupPhaseOns.1	

29.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x02 = 0x02 *Note:* Verify that Max Out indication occurred on Phase 2	Pass/Fail
30.	Set HITL Detector Input 4 and 9 = Off	
31.	DELAY .2 Seconds	
Detector 10 extends Phase 9 when Phase 9 is Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 7 = On	
16.	DELAY .2 Seconds	
17.	Set HITL Detector Input 10 = On	
18.	DELAY .2 Seconds	
19.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
20.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x10 AND phaseStatusGroupPhaseOns.2 ≠ 0x01	
21.	DELAY 1 Second	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WEND *Note:* Wait until controller reaches 5+9.	
24.	Set HITL Detector Input 7 = Off	
25.	DELAY .2 Seconds	
26.	GET ringStatus.1	
27.	WHILE ringStatus.1 AND 0x10 ≠ 0x10 (xxx1xxxx = maxout)	
28.	DELAY 1 Second	
29.	GET ringStatus.1	
30.	WEND *Note:* Wait for Max Out Indication. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
31.	GET phaseStatusGroupPhaseOns.2	

32.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.2 AND 0x01 = 0x01 *Note:* Ensure that Max Out occurred on Phase 9	Pass/Fail
33.	Set HITL Detector Input 10 = Off	
34.	DELAY .2 Seconds	
Teardown		
	PERFORM Detector Teardown – TC020 if not proceeding to another detector operation test case	
Test Case Results		
Tested By:	| Date Tested |	
Test Case Notes:		
Version History:	v1.00 05/09/06 Initial Draft – RDR v1.01 07/05/06 Updated notes – RDR v1.02 07/19/06 Implemented script and proofed – JJ	

Detector 11 Operations

Test Case: **TC011**	*Title*: **Detector 11 Operations** Description: Verifies the operation of Detector 11 to call and extend Phase 2 under specific conditions Constants: Variables: currentExtendValue Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
Setup		
	PERFORM Detector Operations Setup – TC019 if not already done so.	
Detector 11 calls Phase 2 when Phase 2 not Green (In 3+5 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 11 = On	

16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x14 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x02 ≠ 0x02	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x14 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x02 = 0x02 *Note:* Ensure that Phase 2 registers a Vehicle Call during 3+5 Green.	Pass/Fail
23.	Set HITL Detector Input 11 = Off	
24.	DELAY .2 Seconds	
Detector 11 calls Phase 2 when Phase 2 not Green (In 4+5 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 11 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x18 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x02 ≠ 0x02	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x18 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x02 = 0x02 *Note:* Ensure that Phase 2 registers a Vehicle Call during 4+5 Green.	Pass/Fail
23.	Set HITL Detector Input 11 = Off	

24.	DELAY .2 Seconds	
Detector 11 calls Phase 2 when Phase 2 not Green (In 6+12 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 6 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x20 AND phaseStatusGroupPhaseOns.2 ≠ 0x08	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 6+12.	
22.	Set HITL Detector Input 6 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 11 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x08 AND phaseStatusGroupVehCalls.1 AND 0x02 ≠ 0x02	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x08 AND phaseStatusGroupVehCalls.1 AND 0x02 = 0x02 *Note:* Ensure that Phase 2 registers a Vehicle Call during 6+12 Green.	Pass/Fail
32.	Set HITL Detector Input 11 = Off	
33.	DELAY .2 Seconds	

Detector 11 calls Phase 2 when Phase 2 not Green (In 6+11 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 6 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x20 AND phaseStatusGroupPhaseOns.2 ≠ 0x04	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 6+11.	
22.	Set HITL Detector Input 6 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 11 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x04 AND phaseStatusGroupVehCalls.1 AND 0x02 ≠ 0x02	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x04 AND phaseStatusGroupVehCalls.1 AND 0x02 = 0x02 *Note:* Ensure that Phase 2 registers a Vehicle Call during 6+11 Green.	Pass/Fail
32.	Set HITL Detector Input 11 = Off	
33.	DELAY .2 Seconds	
Detector 11 calls Phase 2 when Phase 2 not Green (In 5+9 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	

3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 7 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x10 AND phaseStatusGroupPhaseOns.2 ≠ 0x01	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 5+9.	
22.	Set HITL Detector Input 7 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 11 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x10 AND phaseStatusGroupGreens.2 = 0x01 AND phaseStatusGroupVehCalls.1 AND 0x02 ≠ 0x02	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x10 AND phaseStatusGroupGreens.2 = 0x01 AND phaseStatusGroupVehCalls.1 AND 0x02 = 0x02 *Note:* Ensure that Phase 2 registers a Vehicle Call during 5+9 Green.	Pass/Fail
32.	Set HITL Detector Input 11 = Off	
33.	DELAY .2 Seconds	
Detector 11 calls Phase 2 when Phase 2 not Green (In 1+6 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND	

	Note: Loop until controller rests somewhere.	
6.	Set HITL Detector Input 6 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+6.	
13.	Set HITL Detector Input 6 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 11 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x21 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x02 ≠ 0x02	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x21 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x02 = 0x02 *Note:* Ensure that Phase 2 registers a Vehicle Call during 1+6 Green.	Pass/Fail
23.	Set HITL Detector Input 11 = Off	
24.	DELAY .2 Seconds	
Detector 11 calls Phase 2 when Phase 2 not Green (In 1+7 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 11 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2,	

	phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x41 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x02 ≠ 0x02	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x41 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x02 = 0x02 *Note:* Ensure that Phase 2 registers a Vehicle Call during 1+7 Green.	Pass/Fail
23.	Set HITL Detector Input 11 = Off	
24.	DELAY .2 Seconds	
Detector 11 calls Phase 2 when Phase 2 not Green (In 1+8 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 11 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x81 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x02 ≠ 0x02	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x81 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x02 = 0x02 *Note:* Ensure that Phase 2 registers a Vehicle Call during 1+8 Green.	Pass/Fail
23.	Set HITL Detector Input 11 = Off	
24.	DELAY .2 Seconds	

Detector 11 calls Phase 2 when Phase 2 not Green (In 1+13 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 3 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x01 AND phaseStatusGroupPhaseOns.2 ≠ 0x10	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 1+13.	
22.	Set HITL Detector Input 3 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 11 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x01 AND phaseStatusGroupGreens.2 = 0x10 AND phaseStatusGroupVehCalls.1 AND 0x02 ≠ 0x02	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x01 AND phaseStatusGroupGreens.2 = 0x10 AND phaseStatusGroupVehCalls.1 AND 0x02 = 0x02 *Note:* Ensure that Phase 2 registers a Vehicle Call during 1+13 Green.	Pass/Fail
32.	Set HITL Detector Input 11 = Off	
33.	DELAY .2 Seconds	

Detector 11 extends Phase 2 when Phase 2 is Green until a gap in Detector 11 activity occurs at which time Detector 11 becomes inactive until Phase 2 Yellow		
1.	GET vehicleDetectorExtend.11 and RECORD RESPONSE VALUE in [currentExtendValue]	
2.	SET vehicleDetectorExtend.11 = 40 *Note:* Set Detector 11 extend time = 4 seconds.	
3.	GET ringStatus.1, ringStatus.2	
4.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
5.	DELAY 1 Second	
6.	GET ringStatus.1, ringStatus.2	
7.	WEND *Note:* Loop until controller rests somewhere.	
8.	Set HITL Detector Input 6 = On	
9.	DELAY .2 Seconds	
10.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
11.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
12.	DELAY 1 Second	
13.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
14.	WEND *Note:* Wait until controller reaches 1+6.	
15.	Set HITL Detector Input 6 = Off	
16.	DELAY .2 Seconds	
17.	Set HITL Detector Input 11 = On (To call and then extend Phase 2)	
18.	DELAY .2 Seconds	
19.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
20.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x12 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
21.	DELAY 1 Second	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WEND *Note:* Wait until controller reaches 2+5.	
24.	Set HITL Detector Input 7 = On (Detector 11 will extend Phase 2 when there is an opposing call)	
25.	DELAY .2 Seconds	
26.	GET ringStatus.1	
27.	WHILE ringStatus.1 AND 0x07 ≠ 0x01 (xxxxx001 = extension)	
28.	DELAY .1 Second	
29.	GET ringStatus.1	
30.	WEND *Note:* Wait for indication that extensions are timing. Ring 1 = ***2***, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
31.	GET phaseStatusGroupPhaseOns.1	
32.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x02 = 0x02 *Note:* Verify that the extensions are on Phase 2.	Pass/Fail
33.	Set HITL Detector Input 11 = Off	

34.	DELAY 3 Seconds	
35.	Set HITL Detector Input 11 = On	
36.	DELAY .2 Seconds	
37.	Set HITL Detector Input 11 = Off	
38.	DELAY 2.8 Seconds	
39.	GET ringStatus.1	
40.	VERIFY that RESPONSE VALUE ringStatus.1 AND 0x07 = 0x01 (xxxxx001 = extension) *Note:* Ring 1 = ***2***, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
41.	GET phaseStatusGroupPhaseOns.1	
42.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x02 = 0x02 *Note:* Verify that extensions are still timing on Phase 2 because call was entered < 4 seconds later.	Pass/Fail
43.	Set HITL Detector Input 11 = On	
44.	DELAY .2 Seconds	
45.	Set HITL Detector Input 11 = Off	
46.	DELAY 4.8 Seconds *Note:* Since the time between actuations is now 5 seconds, the timer will gap and therefore disable Detector 11 from putting in any further extensions.	
47.	Set HITL Detector Input 11 = On	
48.	DELAY .2 Seconds	
49.	GET ringStatus.1	
50.	VERIFY that RESPONSE VALUE ringStatus.1 AND 0x07 = 0x04 (xxxxx100 = Yellow Change) *Note:* Verify that the Yellow Change is on Phase 2 because call was entered > 4 seconds later. Ring 1 = ***2***, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
51.	GET phaseStatusGroupPhaseOns.1, ringStatus.1, phaseStatusGroupVehCalls.1	
52.	WHILE (phaseStatusGroupPhaseOns.1 AND 0x02 = 0x02) AND (ringStatus.1 AND 0x07 = 0x04) AND (phaseStatusGroupVehCalls.1 AND 0x02 ≠ 0x02)	
53.	DELAY 1 Second	
54.	GET phaseStatusGroupPhaseOns.1, ringStatus.1, phaseStatusGroupVehCalls.1	
55.	WEND	
56.	VERIFY (phaseStatusGroupPhaseOns.1 AND 0x02 = 0x02) AND (ringStatus.1 AND 0x07 = 0x04) AND (phaseStatusGroupVehCalls.1 AND 0x02 = 0x02) *Note:* Ensure that Phase 2 registers a Vehicle Call during 2 Yellow.	Pass/Fail
57.	Set HITL Detector Input 7 = Off and 11 = Off	
58.	DELAY .2 Seconds	
59.	SET vehicleDetectorExtend.11 = [currentExtendValue], *Note:* Restore original values.	

Teardown				
	PERFORM Detector Teardown – TC020 if not proceeding to another detector operation test case			
Test Case Results				
Tested By:		Date Tested		Pass/Fail
Test Case Notes:	<notes> Test Case 12 does not pass an equivalent			
Version History:	v1.00 05/09/06 Initial Draft – RDR v1.01 07/05/06 Updated notes – RDR v1.02 07/12/06 Implemented script and proofed – JJ			

Detector 12 Operations

Test Case: **TC012**	*Title*: Description: Constants: Variables: Pass/Fail Criteria:	**Detector 12 Operations** Verifies the operation of Detector 12 to call and extend Phase 4 under specific conditions and extend interval 4516B (6+12). currentExtendValue The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*		*Results*
Setup			
	PERFORM Detector Operations Setup – TC019 if not already done so.		
Detector 12 calls Phase 4 when Phase 4 not Green (In 6+12 Green)			
1.	GET ringStatus.1, ringStatus.2		
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03		
3.	DELAY 1 Second		
4.	GET ringStatus.1, ringStatus.2		
5.	WEND *Note:* Loop until controller rests somewhere.		
6.	Set HITL Detector Input 4 = On		
7.	DELAY .2 Seconds		
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2		
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0X00		
10.	DELAY 1 Second		
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2		
12.	WEND *Note:* Wait until controller reaches 4+5.		
13.	Set HITL Detector Input 4 = Off		
14.	DELAY .2 Seconds		
15.	GET ringStatus.1, ringStatus.2		
16.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03		

17.	DELAY 1 Second	
18.	GET ringStatus.1, ringStatus.2	
19.	WEND *Note:* Wait for 4+5 Green Rest.	
20.	Set HITL Detector Input 6 = On	
21.	DELAY .2 Seconds	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x20 AND phaseStatusGroupPhaseOns.2 ≠ 0x08	
24.	DELAY 1 Second	
25.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
26.	WEND *Note:* Wait until controller reaches 6+12.	
27.	Set HITL Detector Input 6 = Off	
28.	DELAY .2 Seconds	
29.	Set HITL Detector Input 12 = On	
30.	DELAY .2 Seconds	
31.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
32.	WHILE phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x08 AND phaseStatusGroupVehCalls.1 AND 0x08 ≠ 0x08	
33.	DELAY 1 Second	
34.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
35.	WEND	
36.	VERIFY phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x08 AND phaseStatusGroupVehCalls.1 AND 0x08 = 0x08 *Note:* Ensure that Phase 4 registers a Vehicle Call during 6+12 Green.	Pass/Fail
37.	Set HITL Detector Input 12 = Off	
38.	DELAY .2 Seconds	
Detector 12 calls Phase 4 when Phase 4 not Green (In 6+11 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	

12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	GET ringStatus.1, ringStatus.2	
16.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
17.	DELAY 1 Second	
18.	GET ringStatus.1, ringStatus.2	
19.	WEND *Note:* Wait for 3+5 Green Rest.	
20.	Set HITL Detector Input 6 = On	
21.	DELAY .2 Seconds	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x20 AND phaseStatusGroupPhaseOns.2 ≠ 0x04	
24.	DELAY 1 Second	
25.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
26.	WEND *Note:* Wait until controller reaches 6+11.	
27.	Set HITL Detector Input 6 = Off	
28.	DELAY .2 Seconds	
29.	Set HITL Detector Input 12 = On	
30.	DELAY .2 Seconds	
31.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
32.	WHILE phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x04 AND phaseStatusGroupVehCalls.1 AND 0x08 ≠ 0x08	
33.	DELAY 1 Second	
34.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
35.	WEND	
36.	VERIFY phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x04 AND phaseStatusGroupVehCalls.1 AND 0x08 = 0x08 *Note:* Ensure that Phase 4 registers a Vehicle Call during 6+11 Green.	Pass/Fail
37.	Set HITL Detector Input 12 = Off	
38.	DELAY .2 Seconds	
Detector 12 calls Phase 4 when Phase 4 not Green (In 5+9 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	

9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	GET ringStatus.1, ringStatus.2	
16.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
17.	DELAY 1 Second	
18.	GET ringStatus.1, ringStatus.2	
19.	WEND *Note:* Wait for 3+5 Green Rest.	
20.	Set HITL Detector Input 7 = On	
21.	DELAY .2 Seconds	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x10 AND phaseStatusGroupPhaseOns.2 ≠ 0x01	
24.	DELAY 1 Second	
25.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
26.	WEND *Note:* Wait until controller reaches 5+9.	
27.	Set HITL Detector Input 7 = Off	
28.	DELAY .2 Seconds	
29.	Set HITL Detector Input 12 = On	
30.	DELAY .2 Seconds	
31.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
32.	WHILE phaseStatusGroupGreens.1 = 0x10 AND phaseStatusGroupGreens.2 = 0x01 AND phaseStatusGroupVehCalls.1 AND 0x08 ≠ 0x08	
33.	DELAY 1 Second	
34.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
35.	WEND	
36.	VERIFY phaseStatusGroupGreens.1 = 0x10 AND phaseStatusGroupGreens.2 = 0x01 AND phaseStatusGroupVehCalls.1 AND 0x08 = 0x08 *Note:* Ensure that Phase 4 registers a Vehicle Call during 5+9 Green.	Pass/Fail
37.	Set HITL Detector Input 12 = Off	
38.	DELAY .2 Seconds	
Detector 12 calls Phase 4 when Phase 4 not Green (In 1+6 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	

6.	Set HITL Detector Input 6 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+6.	
13.	Set HITL Detector Input 6 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 12 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x21 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x08 ≠ 0x08	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x21 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x08 = 0x08 *Note:* Ensure that Phase 4 registers a Vehicle Call during 1+6 Green.	Pass/Fail
23.	Set HITL Detector Input 12 = Off	
24.	DELAY .2 Seconds	
Detector 12 calls Phase 4 when Phase 4 not Green (In 1+7 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 12 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	

18.	WHILE phaseStatusGroupGreens.1 = 0x41 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x08 ≠ 0x08	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x41 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x08 = 0x08 *Note:* Ensure that Phase 4 registers a Vehicle Call during 1+7 Green.	Pass/Fail
23.	Set HITL Detector Input 12 = Off	
24.	DELAY .2 Seconds	
Detector 12 calls Phase 4 when Phase 4 not Green (In 1+8 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 12 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x81 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x08 ≠ 0x08	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x81 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x08 = 0x08 *Note:* Ensure that Phase 4 registers a Vehicle Call during 1+8 Green.	Pass/Fail
23.		
24.	Set HITL Detector Input 12 = Off	
25.	DELAY .2 Seconds	
Detector 12 calls Phase 4 when Phase 4 not Green (In 2+16 Green)		

1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	GET ringStatus.1, ringStatus.2	
16.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
17.	DELAY 1 Second	
18.	GET ringStatus.1, ringStatus.2	
19.	WEND *Note:* Wait for 1+8 Green Rest.	
20.	Set HITL Detector Input 2 = On	
21.	DELAY .2 Seconds	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x02 AND phaseStatusGroupPhaseOns.2 ≠ 0x80	
24.	DELAY 1 Second	
25.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
26.	WEND *Note:* Wait until controller reaches 2+16.	
27.	Set HITL Detector Input 2 = Off	
28.	DELAY .2 Seconds	
29.	Set HITL Detector Input 12 = On	
30.	DELAY .2 Seconds	
31.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
32.	WHILE phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x80 AND phaseStatusGroupVehCalls.1 AND 0x08 ≠ 0x08	
33.	DELAY 1 Second	
34.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
35.	WEND	

36.	VERIFY phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x80 AND phaseStatusGroupVehCalls.1 AND 0x08 = 0x08 *Note:* Ensure that Phase 4 registers a Vehicle Call during 2+16 Green.	Pass/Fail
37.	Set HITL Detector Input 12 = Off	
38.	DELAY .2 Seconds	
Detector 12 calls Phase 4 when Phase 4 not Green (In 2+15 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	GET ringStatus.1, ringStatus.2	
16.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
17.	DELAY 1 Second	
18.	GET ringStatus.1, ringStatus.2	
19.	WEND *Note:* Wait for 1+7 Green Rest.	
20.	Set HITL Detector Input 2 = On	
21.	DELAY .2 Seconds	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x02 AND phaseStatusGroupPhaseOns.2 ≠ 0x40	
24.	DELAY 1 Second	
25.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
26.	WEND *Note:* Wait until controller reaches 2+15.	
27.	Set HITL Detector Input 2 = Off	
28.	DELAY .2 Seconds	
29.	Set HITL Detector Input 12 = On	
30.	DELAY .2 Seconds	
31.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
32.	WHILE phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x40 AND phaseStatusGroupVehCalls.1 AND 0x08 ≠ 0x08	

33.	DELAY 1 Second	
34.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
35.	WEND	
36.	VERIFY phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x40 AND phaseStatusGroupVehCalls.1 AND 0x08 = 0x08 *Note:* Ensure that Phase 4 registers a Vehicle Call during 2+15 Green.	Pass/Fail
37.	Set HITL Detector Input 12 = Off	
38.	DELAY .2 Seconds	
Detector 12 calls Phase 4 when Phase 4 not Green (In 1+13 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	GET ringStatus.1, ringStatus.2	
16.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
17.	DELAY 1 Second	
18.	GET ringStatus.1, ringStatus.2	
19.	WEND *Note:* Wait for 1+8 Green Rest.	
20.	Set HITL Detector Input 3 = On	
21.	DELAY .2 Seconds	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x01 AND phaseStatusGroupPhaseOns.2 ≠ 0x10	
24.	DELAY 1 Second	
25.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
26.	WEND *Note:* Wait until controller reaches 1+13.	
27.	Set HITL Detector Input 3 = Off	
28.	DELAY .2 Seconds	
29.	Set HITL Detector Input 12 = On	
30.	DELAY .2 Seconds	

31.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
32.	WHILE phaseStatusGroupGreens.1 = 0x01 AND phaseStatusGroupGreens.2 = 0x10 AND phaseStatusGroupVehCalls.1 AND 0x08 ≠ 0x08	
33.	DELAY 1 Second	
34.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
35.	WEND	
36.	VERIFY phaseStatusGroupGreens.1 = 0x01 AND phaseStatusGroupGreens.2 = 0x10 AND phaseStatusGroupVehCalls.1 AND 0x08 = 0x08 *Note:* Ensure that Phase 4 registers a Vehicle Call during 1+13 Green.	Pass/Fail
37.	Set HITL Detector Input 12 = Off	
38.	DELAY .2 Seconds	
Detector 12 calls Phase 4 when Phase 4 not Green (In 2+5 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 2 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x12 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 2+5.	
13.	Set HITL Detector Input 2 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 12 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x12 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x08 ≠ 0x08	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x12 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x08 = 0x08 *Note:* Ensure that Phase 4 registers a Vehicle Call during 2+5 Green.	Pass/Fail
23.	Set HITL Detector Input 12 = Off	

24.	DELAY .2 Seconds	
Detector 12 calls Phase 4 when Phase 4 not Green (In 3+5 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 12 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x14 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x08 ≠ 0x08	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x14 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x08 = 0x08 *Note:* Ensure that Phase 4 registers a Vehicle Call during 3+5 Green.	Pass/Fail
23.	Set HITL Detector Input 12 = Off	
24.	DELAY .2 Seconds	
Detector 12 extends Phase 4 until Phase 6 call exists and Detector 12 gaps and Phase 6 call continues		
1.	GET vehicleDetectorExtend.12 and RECORD RESPONSE in [currentExtendValue]	
2.	SET vehicleDetectorExtend.12 = 40 *Note:* Set Detector 12 extend time = 4 seconds so that actuations less than 4 seconds apart keep the phase extending.	
3.	GET ringStatus.1, ringStatus.2	
4.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
5.	DELAY 1 Second	
6.	GET ringStatus.1, ringStatus.2	
7.	WEND *Note:* Loop until controller rests somewhere.	

8.	Set HITL Detector Input 8 = On	
9.	DELAY .2 Seconds	
10.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
11.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
12.	DELAY 1 Second	
13.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
14.	WEND *Note:* Wait until controller reaches 1+8.	
15.	Set HITL Detector Input 8 = Off	
16.	DELAY .2 Seconds	
17.	Set HITL Detector Input 12 = On (To call and then extend Phase 4	
18.	DELAY .2 Seconds	
19.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
20.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
21.	DELAY 1 Second	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WEND *Note:* Wait until controller reaches 4+5.	
24.	Set HITL Detector Input 6 = On (Detector 12 will extend Phase 4 when there is an opposing call)	
25.	DELAY .2 Seconds *Note:* This should get Phase 4 to start timing extensions and sets up conditional logic.	
26.	GET ringStatus.1	
27.	WHILE ringStatus.1 AND 0x07 ≠ 0x01 (xxxxx001 = extension)	
28.	DELAY .1 Second	
29.	GET ringStatus.1	
30.	WEND *Note:* Wait for indication that extensions are timing. Ring 1 = 2, 3, ***4***, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
31.	GET phaseStatusGroupPhaseOns.1	
32.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x08 = 0x08 *Note:* Verify that the extensions are on Phase 4.	Pass/Fail
33.	Set HITL Detector Input 12 = Off	
34.	DELAY 3 Seconds	
35.	Set HITL Detector Input 12 = On	
36.	DELAY .2 Seconds	
37.	Set HITL Detector Input 12 = Off	
38.	DELAY 2.8 Seconds	
39.	GET ringStatus.1	
40.	VERIFY that RESPONSE VALUE ringStatus.1 AND 0x07 = 0x01 (xxxxx001 = extension) *Note:* Ring 1 = 2, 3, ***4***, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	Pass/Fail

41.	GET phaseStatusGroupPhaseOns.1	
42.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x08 = 0x08 *Note:* Verify that extensions are still timing on Phase 4 because call was entered < 4 seconds later.	Pass/Fail
43.	Set HITL Detector Input 12 = On	
44.	DELAY .2 Seconds	
45.	Set HITL Detector Input 12 = Off	
46.	DELAY 2.8 Seconds	
47.	GET ringStatus.1	
48.	VERIFY that RESPONSE VALUE ringStatus.1 AND 0x07 = 0x01 (xxxxx001 = extension) *Note:* Ring 1 = 2, 3, ***4***, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	Pass/Fail
49.	GET phaseStatusGroupPhaseOns.1	
50.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x08 = 0x08 *Note:* Verify that extensions are still timing on Phase 4 because call was entered < 4 seconds later.	Pass/Fail
51.	Set HITL Detector Input 12 = On	
52.	DELAY .2 Seconds	
53.	Set HITL Detector Input 12 = Off	
54.	DELAY 4.8 Seconds *Note:* Since the time between actuations is now 5 seconds, the timer will gap and therefore disable Detector 12 from putting in any further extensions.	
55.	Set HITL Detector Input 12 = On	
56.	DELAY .2 Seconds	
57.	GET ringStatus.1	
58.	VERIFY that RESPONSE VALUE ringStatus.1 AND 0x07 ≠ 0x01 (xxxxx001 = extension) *Note:* Verify that Phase 4 is no longer timing extensions. Ring 1 = 2, 3, ***4***, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	Pass/Fail
59.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupVehCalls.1	
60.	WHILE (phaseStatusGroupPhaseOns.1 AND 0x08 = 0x08) AND (phaseStatusGroupVehCalls.1 AND 0x08 ≠ 0x08)	
61.	DELAY 1 Second	
62.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupVehCalls.1	
63.	WEND	
64.	VERIFY (phaseStatusGroupPhaseOns.1 AND 0x08 = 0x08) AND (phaseStatusGroupVehCalls.1 AND 0x08 = 0x08) *Note:* Ensure that Phase 4 registers a Vehicle Call.	Pass/Fail
65.	Set HITL Detector Input 6 = Off and 12 = Off	
66.	DELAY .2 Seconds	
67.	POST-CONDITION The Detector 12 vehicleDetectorExtend is still set to 4 seconds	

Detector 12 extends Phase 4 until Phase 6 call exists and Detector 12 gaps but not if Phase 6 call disappears		
	PRE-CONDITION This procedure assumes that Detector 12 vehicleDetectorExtend is still set to 4 seconds	
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 12 = On (To call and then extend Phase 4)	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 4+5.	
22.	Set HITL Detector Input 6 = On	
23.	DELAY .2 Seconds *Note:* Detector 12 will extend Phase 4 when there is an opposing call.	
24.	GET ringStatus.1	
25.	WHILE ringStatus.1 AND 0x07 ≠ 0x01 (xxxxx001 = extension)	
26.	DELAY .1 Second	
27.	GET ringStatus.1	
28.	WEND *Note:* Wait for indication that extensions are timing. Ring 1 = 2, 3, ***4***, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
29.	GET phaseStatusGroupPhaseOns.1	
30.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x08 = 0x08 *Note:* Verify that the extensions are timing on Phase 4.	Pass/Fail
31.	Set HITL Detector Input 12 = Off	
32.	DELAY 3 Seconds	

33.	Set HITL Detector Input 12 = On	
34.	DELAY .2 Seconds	
35.	Set HITL Detector Input 12 = Off	
36.	DELAY 2.8 Seconds	
37.	GET ringStatus.1	
38.	VERIFY that RESPONSE VALUE ringStatus.1 AND 0x07 = 0x01 (xxxxx001 = extension) Ring 1 = 2, 3, ***4***, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	Pass/Fail
39.	GET phaseStatusGroupPhaseOns.1	
40.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x08 = 0x08 *Note:* Verify that extensions are still timing is on Phase 4.	Pass/Fail
41.	Set HITL Detector Input 12 = On	
42.	DELAY .2 Seconds	
43.	Set HITL Detector Input 12 = Off	
44.	DELAY 2.8 Seconds	
45.	GET ringStatus.1	
46.	VERIFY that RESPONSE VALUE ringStatus.1 AND 0x07 = 0x01 (xxxxx001 = extension) *Note:* Ring 1 = 2, 3, ***4***, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	Pass/Fail
47.	GET phaseStatusGroupPhaseOns.1	
48.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x08 = 0x08 *Note:* Verify that extensions are still timing on Phase 4.	Pass/Fail
49.	Set HITL Detector Input 12 = On	
50.	DELAY .2 Seconds	
51.	Set HITL Detector Input 12 = Off, 4 = On, 6 = Off and 7 = On *Note:* Setting Detector 4 on will keep extending Phase 4 irrespective of Detector 12, no call on Phase 6 will reset Detector 12 gap function, and the call on Phase 7 will enable extensions to time.	
52.	DELAY 4.8 Seconds *Note:* This would have the effect of allowing the extend timer to gap and therefore disable detector 12 but since Phase 6 no longer has a call, another activation of Detector 12 will continue to extend Phase 4.	
53.	Set HITL Detector Input 12 = On and 4 Off	
54.	DELAY .2 Seconds	
55.	Set HITL Detector Input 12 = Off	
56.	DELAY .2 Seconds	
57.	GET ringStatus.1	
58.	VERIFY that RESPONSE VALUE ringStatus.1 AND 0x07 = 0x01 (xxxxx001 = extension)	Pass/Fail
59.	GET phaseStatusGroupPhaseOns.1	

60.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x08 = 0x08 *Note:* Verify that extensions are still timing on Phase 4 because even though call was > 4 seconds later, the absence of call on phase 6 allowed phase 4 to continue.	Pass/Fail
61.	Set HITL Detector Input 7 = Off	
62.	DELAY .2 Seconds	
63.	SET vehicleDetectorExtend.12 = [currentExtendValue] *Note:* Restore original values.	
Detector 12 extends Phase 12 when Phase 12 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 12 = On	
16.	DELAY .2 Seconds	
17.	Set HITL Detector Input 6 = On	
18.	DELAY .2 Seconds	
19.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
20.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x20 AND phaseStatusGroupPhaseOns.2 ≠ 0X08	
21.	DELAY 1 Second	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WEND *Note:* Wait until controller reaches 6+12.	
24.	Set HITL Detector Input 6 = Off	
25.	DELAY .2 Seconds	
26.	GET ringStatus.1	
27.	WHILE ringStatus.1 AND 0x10 ≠ 0x10 (xxx1xxxx = maxout)	
28.	DELAY 1 Second	
29.	GET ringStatus.1	

30.	WEND *Note:* Wait for Max Out Indication. Ring 1 = 2, 3, 4, 9, 11, ***12***, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
31.	GET phaseStatusGroupPhaseOns.2	
32.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.2 AND 0x08 = 0x08 *Note:* Ensure that Max Out occurred on Phase 12.	Pass/Fail
33.	Set HITL Detector Input 12 = Off	
34.	DELAY .2 Seconds	
Teardown		
	PERFORM Detector Teardown – TC020 if not proceeding to another detector operation test case	
	Test Case Results	
Tested By:	| Date Tested |	Pass/Fail
Test Case Notes:	<notes> 1. The section on “Detector 12 extends Phase 4 until Phase 6 call exists and Detector 12 gaps and Phase 6 call continues“ is not the equivalent as the last steps in TC011 and TC015. This may be due to a timing issue.	
Version History:	v1.00 05/09/06 Initial Draft – RDR v1.01 07/05/06 Updated notes – RDR v1.02 07/31/06 Implemented script and proofed – JJ	

Detector 13 Operations

Test Case: **TC013**	*Title*: **Detector 13 Operations** Description: Verifies the operation of Detector 13 to call Phase 2 under specific conditions, extend Phase 6 under specific conditions, and extend intervals 1625B, 1635B, 1645B, 1735B, 1745B, 1835B, and 1845B. Constants: Variables: Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
Setup		
	PERFORM Detector Operations Setup – TC019 if not already done so.	
Detector 13 No Call on Phase 2 when Overlap B is Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 3 = On	

7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY 2 Seconds	
15.	GET overlapStatusGroupGreens.1 *Note:* overlapStatusGroupGreens is optional and a GET may return a noSuchName. This also assumes that Overlap B = 5+6+X.	
16.	IF RESPONSE ERROR = noError THEN	
17.	VERIFY that RESPONSE VALUE overlapStatusGroupGreens.1 AND 0x02 = 0x02 *Note:* Verifies that Overlap B = Green.	Pass/Fail
18.	END IF	
19.	Set HITL Detector Input 13 = On	
20.	DELAY 2 Seconds	
21.	GET phaseStatusGroupVehCalls.1, phaseStatusGroupVehCalls.2	
22.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 = 0x00 AND phaseStatusGroupVehCalls.2 = 0x00 *Note:* Checks for no call on Phase 2 or anywhere	Pass/Fail
23.	Set HITL Detector Input 13 = Off	
24.	DELAY .2 Seconds	
Detector 13 No Call on Phase 2 when Overlap B is not Green and a Call on Phase 3		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	

15.	GET overlapStatusGroupGreens.1 *Note:* overlapStatusGroupGreens is optional and a GET may return a noSuchName. This also assumes that Overlap B = 5+6+X.	
16.	IF RESPONSE ERROR = noError THEN	
17.	VERIFY that RESPONSE VALUE overlapStatusGroupGreens.1 AND 0x02 = 0x00 *Note:* Verifies that Overlap B = NOT Green.	Pass/Fail
18.	END IF	
19.	Set HITL Detector Input 3 = On	
20.	DELAY 2 Seconds	
21.	Set HITL Detector Input 13 = On	
22.	DELAY 2 Seconds	
23.	GET phaseStatusGroupVehCalls.1	
24.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 AND 0x02 = 0x00 *Note:* Checks for no call on Phase 2	Pass/Fail
25.	Set HITL Detector Input 3 = Off	
26.	DELAY .2 Seconds	
27.	Set HITL Detector Input 13 = Off	
28.	DELAY .2 Seconds	
Detector 13 No Call on Phase 2 when Overlap B is not Green and a Call on Phase 4		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	GET overlapStatusGroupGreens.1 *Note:* overlapStatusGroupGreens is optional and a GET may return a noSuchName. This also assumes that Overlap B = 5+6+X.	
16.	IF RESPONSE ERROR = noError THEN	
17.	VERIFY that RESPONSE VALUE overlapStatusGroupGreens.1 AND 0x02 = 0x00 *Note:* Verifies that Overlap B = NOT Green.	
18.	END IF	

19.	Set HITL Detector Input 4 = On	
20.	DELAY 2 Seconds	
21.	Set HITL Detector Input 13 = On	
22.	DELAY 2 Seconds	
23.	GET phaseStatusGroupVehCalls.1	
24.	VERIFY that RESPONSE VALUE AND 0x02 = 0 *Note:* Checks for no call on Phase 2	Pass/Fail
25.	Set HITL Detector Input 13 = Off	
26.	DELAY .2 Seconds	
27.	Set HITL Detector Input 4 = Off	
28.	DELAY .2 Seconds	
Detector 13 calls Phase 2 when Overlap B is not Green and no Calls on Phase 3 or Phase 4		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	GET overlapStatusGroupGreens.1 *Note:* overlapStatusGroupGreens is optional and a GET may return a noSuchName. This also assumes that Overlap B = 5+6+X.	
16.	IF RESPONSE ERROR = noError THEN	
17.	VERIFY that RESPONSE VALUE overlapStatusGroupGreens.1 AND 0x02 = 0x00 *Note:* Verifies that Overlap B = NOT Green.	Pass/Fail
18.	END IF	
19.	Set HITL Detector Input 13 = On	
20.	DELAY 2 Seconds	
21.	GET phaseStatusGroupVehCalls.1	
22.	VERIFY that RESPONSE VALUE AND 0x02 = 0x02 *Note:* Checks for call on Phase 2	Pass/Fail
23.	Set HITL Detector Input 13 = Off	
24.	DELAY .2 Seconds	

Detector 13 does not extend Phase 6 when Phase 6 is Green and there are not Calls on Phase 7 or Phase 8		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 6 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+6.	
13.	Set HITL Detector Input 6 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 13 = On *Note:* To possible extend	
16.	DELAY .2 Seconds	
17.	Set HITL Detector Input 6 = On *Note:* To force extensions	
18.	DELAY .2 Seconds	
19.	Set HITL Detector Input 2 = On *Note:* When opposing call exists	
20.	DELAY .2 Seconds	
21.	Set HITL Detector Input 6 = Off *Note:* To check whether possible extensions are due to Detector 13.	
22.	DELAY .2 Seconds	
23.	GET ringStatus.2	
24.	WHILE ringStatus.1 AND 0x08 ≠ 0x08 (xxxx1xxx = gap out)	
25.	DELAY 1 Second	
26.	GET ringStatus.2	
27.	WEND *Note:* Wait for Gap Out indication on Ring 2. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
28.	GET phaseStatusGroupPhaseOns.1	
29.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x20 = 0x20 *Note:* Verify that Gap Out indication occurred on Phase 6.	Pass/Fail
30.	Set HITL Detector Input 2 and 13 = Off	
31.	DELAY .2 Seconds	

Detector 13 extends Phase 6 when Phase 6 is Green and there is a Call on Phase 7		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 6 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+6.	
13.	Set HITL Detector Input 6 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 13 = On *Note:* To possible extend	
16.	DELAY .2 Seconds	
17.	Set HITL Detector Input 6 = On *Note:* To force extensions	
18.	DELAY .2 Seconds	
19.	Set HITL Detector Input 7 = On *Note:* When conflicting call exists and satisfy conditions	
20.	DELAY .2 Seconds	
21.	Set HITL Detector Input 6 = Off *Note:* To check whether possible extensions are due to Detector 13.	
22.	DELAY .2 Seconds	
23.	GET ringStatus.2	
24.	WHILE ringStatus.2 AND 0x10 ≠ 0x10 (xxx1xxxx = maxout)	
25.	DELAY 1 Second	
26.	GET ringStatus.2	
27.	WEND *Note:* Wait for Max Out Indication on Ring 2. Ring 1 = 2, 3, 4, 9, 11, & 12 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
28.	GET phaseStatusGroupPhaseOns.1	
29.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x20 = 0x20 *Note:* Verify that Max Out indication occurred on Phase 6.	Pass/Fail
30.	Set HITL Detector Input 7 and 13 = Off	
31.	DELAY .2 Seconds	

Detector 13 extends Phase 6 when Phase 6 is Green and there is a call on Phase 8		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 6 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+6.	
13.	Set HITL Detector Input 6 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 13 = On *Note:* To possible extend	
16.	DELAY .2 Seconds	
17.	Set HITL Detector Input 6 = On *Note:* To force extensions	
18.	DELAY .2 Seconds	
19.	Set HITL Detector Input 8 = On *Note:* When opposing call exists	
20.	DELAY .2 Seconds	
21.	Set HITL Detector Input 6 = Off *Note:* To check whether possible extensions are due to Detector 13.	
22.	DELAY .2 Seconds	
23.	GET ringStatus.2	
24.	WHILE ringStatus.1 AND 0x10 ≠ 0x10 (xxx1xxxx = maxout)	
25.	DELAY 1 Second	
26.	GET ringStatus.2	
27.	WEND *Note:* Wait for Max Out Indication on Ring 2. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
28.	GET phaseStatusGroupPhaseOns.1	
29.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x20= 0x20 *Note:* Verify that Max Out indication occurred on Phase 6.	Pass/Fail
30.	Set HITL Detector Input 8 and 13 = Off	
31.	DELAY .2 Seconds	

Detector 13 extends Phase 13 when Phase 13 is Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 3 = On	
16.	DELAY .2 Seconds	
17.	Set HITL Detector Input 13 = On	
18.	DELAY .2 Seconds	
19.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
20.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x01 AND phaseStatusGroupPhaseOns.2 ≠ 0x10	
21.	DELAY 1 Second	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WEND *Note:* Wait until controller reaches 1+13.	
24.		
25.	Set HITL Detector Input 3 = Off	
26.	DELAY .2 Seconds	
27.	GET ringStatus.2	
28.	WHILE ringStatus.2 AND 0x10 ≠ 0x10 (xxx1xxxx = maxout)	
29.	DELAY 1 Second	
30.	GET ringStatus.2	
31.	WEND *Note:* Wait for Max Out Indication on Ring 2. Ring 1 = 2, 3, 4, 9, 11, & 12 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
32.	GET phaseStatusGroupPhaseOns.2	
33.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.2 AND 0x10 = 0x10 *Note:* Ensure that Max Out occurred on Phase 13.	Pass/Fail
34.	Set HITL Detector Input 13 = Off	
35.	DELAY .2 Seconds	

Detector 13 Teardown			
	PERFORM Detector Teardown – TC020 if not proceeding to another detector operation test case		
Test Case Results			
Tested By:		Date Tested	
Test Case Notes:			
Version History:	v1.00 04/12/06 Initial Draft – RDR v1.01 07/05/06 Deleted strikethroughs, moved initial detector turnoff point, corrected detector number in some cases, and updated notes – RDR v1.02 07/13/06 Implemented script and proofed – JJ		

Detector 14 Operations

Test Case: **TC014**	*Title*: Description: Constants: Variables: Pass/Fail Criteria:	**Detector 14 Operations** Verifies the operation of Detector 14 to call Phase 2 under specific conditions, extend Phase 6 under specific conditions, and extend intervals 1625B, 1635B, 1645B, 1735B, 1745B, 1835B, and 1845B. The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*		*Results*
Setup			
	PERFORM Detector Operations Setup – TC019 if not already done so.		
Detector 14 No Call on Phase 2 when Overlap B is Green			
1.	GET ringStatus.1, ringStatus.2		
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03		
3.	DELAY 1 Second		
4.	GET ringStatus.1, ringStatus.2		
5.	WEND *Note:* Loop until controller rests somewhere.		
6.	Set HITL Detector Input 3 = On		
7.	DELAY .2 Seconds		
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2		
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00		
10.	DELAY 1 Second		
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2		
12.	WEND *Note:* Wait until controller reaches 3+5.		
13.	Set HITL Detector Input 3 = Off		
14.	DELAY 2 Seconds		
15.	GET overlapStatusGroupGreens.1 *Note:* overlapStatusGroupGreens is optional and a GET may return a noSuchName. This also assumes that Overlap B = 5+6+X.		

16.	IF RESPONSE ERROR = noError THEN	
17.	VERIFY that RESPONSE VALUE overlapStatusGroupGreens.1 AND 0x02 = 0x02 *Note:* Verifies that Overlap B = Green.	Pass/Fail
18.	END IF	
19.	Set HITL Detector Input 14 = On	
20.	DELAY 2 Seconds	
21.	GET phaseStatusGroupVehCalls.1, phaseStatusGroupVehCalls.2	
22.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 = 0x00 AND phaseStatusGroupVehCalls.2 = 0x00 *Note:* Checks for no call on Phase 2 or anywhere	Pass/Fail
23.	Set HITL Detector Input 14 = Off	
24.	DELAY .2 Seconds	
Detector 14 No Call on Phase 2 when Overlap B is not Green and a Call on Phase 3		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	GET overlapStatusGroupGreens.1 *Note:* overlapStatusGroupGreens is optional and a GET may return a noSuchName. This also assumes that Overlap B = 5+6+X.	
16.	IF RESPONSE ERROR = noError THEN	
17.	VERIFY that RESPONSE VALUE overlapStatusGroupGreens.1 AND 0x02 = 0x00 *Note:* Verifies that Overlap B = NOT Green.	Pass/Fail
18.	END IF	
19.	Set HITL Detector Input 3 = On	
20.	DELAY 2 Seconds	
21.	Set HITL Detector Input 14 = On	
22.	DELAY 2 Seconds	
23.	GET phaseStatusGroupVehCalls.1	

24.	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 AND 0x02 = 0x00 *Note:* Checks for no call on Phase 2	Pass/Fail
25.	Set HITL Detector Input 3 = Off	
26.	DELAY .2 Seconds	
27.	Set HITL Detector Input 14 = Off	
28.	DELAY .2 Seconds	
Detector 14 No Call on Phase 2 when Overlap B is not Green and a Call on Phase 4		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	GET overlapStatusGroupGreens.1 *Note:* overlapStatusGroupGreens is optional and a GET may return a noSuchName. This also assumes that Overlap B = 5+6+X.	
16.	IF RESPONSE ERROR = noError THEN	
17.	VERIFY that RESPONSE VALUE overlapStatusGroupGreens.1 AND 0x02 = 0x00 *Note:* Verifies that Overlap B = NOT Green.	Pass/Fail
18.	END IF	
19.	Set HITL Detector Input 4 = On	
20.	DELAY 2 Seconds	
21.	Set HITL Detector Input 14 = On	
22.	DELAY 2 Seconds	
23.	GET phaseStatusGroupVehCalls.1	
24.	VERIFY that RESPONSE VALUE AND 0x02 = 0 *Note:* Checks for no call on Phase 2	Pass/Fail
25.	Set HITL Detector Input 14 = Off	
26.	DELAY .2 Seconds	
27.	Set HITL Detector Input 4 = Off	
28.	DELAY .2 Seconds	

Detector 14 calls Phase 2 when Overlap B is not Green and no Calls on Phase 3 or Phase 4		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	GET overlapStatusGroupGreens.1 *Note:* overlapStatusGroupGreens is optional and a GET may return a noSuchName. This also assumes that Overlap B = 5+6+X.	
16.	IF RESPONSE ERROR = noError THEN	
17.	VERIFY that RESPONSE VALUE overlapStatusGroupGreens.1 AND 0x02 = 0x00 *Note:* Verifies that Overlap B = NOT Green	Pass/Fail
18.	END IF	
19.	Set HITL Detector Input 14 = On	
20.	DELAY 2 Seconds	
21.	GET phaseStatusGroupVehCalls.1	
22.	VERIFY that RESPONSE VALUE AND 0x02 = 0x02 *Note:* Checks for call on Phase 2	Pass/Fail
23.	Set HITL Detector Input 14 = Off	
24.	DELAY .2 Seconds	
Detector 14 does not extend Phase 6 when Phase 6 is Green and there are not Calls on Phase 7 or Phase 8		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 6 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	

11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+6.	
13.	Set HITL Detector Input 6 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 14 = On *Note:* To possible extend	
16.	DELAY .2 Seconds	
17.	Set HITL Detector Input 6 = On *Note:* To force extensions	
18.	DELAY .2 Seconds	
19.	Set HITL Detector Input 2 = On *Note:* When opposing call exists	
20.	DELAY .2 Seconds	
21.	Set HITL Detector Input 6 = Off *Note:* To check whether possible extensions are due to Detector 14.	
22.	DELAY .2 Seconds	
23.	GET ringStatus.2	
24.	WHILE ringStatus.1 AND 0x08 ≠ 0x08 (xxxx1xxx = gap out)	
25.	DELAY 1 Second	
26.	GET ringStatus.2	
27.	WEND *Note:* Wait for Gap Out indication on Ring 2. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
28.	GET phaseStatusGroupPhaseOns.1	
29.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x20 = 0x20 *Note:* Verify that Gap Out indication occurred on Phase 6.	Pass/Fail
30.	Set HITL Detector Input 2 and 14 = Off	
31.	DELAY .2 Seconds	
Detector 14 extends Phase 6 when Phase 6 is Green and there is a Call on Phase 7		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 6 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	

11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+6.	
13.	Set HITL Detector Input 6 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 14 = On *Note:* To possible extend	
16.	DELAY .2 Seconds	
17.	Set HITL Detector Input 6 = On *Note:* To force extensions	
18.	DELAY .2 Seconds	
19.	Set HITL Detector Input 7 = On *Note:* When conflicting call exists and satisfies conditions	
20.	DELAY .2 Seconds	
21.	Set HITL Detector Input 6 = Off *Note:* To check whether possible extensions are due to Detector 14.	
22.	DELAY .2 Seconds	
23.	GET ringStatus.2	
24.	WHILE ringStatus.2 AND 0x10 ≠ 0x10 (xxx1xxxx = maxout)	
25.	DELAY 1 Second	
26.	GET ringStatus.2	
27.	WEND *Note:* Wait for Max Out Indication on Ring 2. Ring 1 = 2, 3, 4, 9, 11, & 12 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
28.	GET phaseStatusGroupPhaseOns.1	
29.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x20 = 0x20 *Note:* Verify that Max Out indication occurred on Phase 6.	Pass/Fail
30.	Set HITL Detector Input 7 and 14 = Off	
31.	DELAY .2 Seconds	
Detector 14 extends Phase 6 when Phase 6 is Green and there is a call on Phase 8		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 6 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	

11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+6.	
13.	Set HITL Detector Input 6 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 14 = On *Note:* To possible extend	
16.	DELAY .2 Seconds	
17.	Set HITL Detector Input 6 = On *Note:* To force extensions	
18.	DELAY .2 Seconds	
19.	Set HITL Detector Input 8 = On *Note:* When opposing call exists	
20.	DELAY .2 Seconds	
21.	Set HITL Detector Input 6 = Off *Note:* To check whether possible extensions are due to Detector 14.	
22.	DELAY .2 Seconds	
23.	GET ringStatus.2	
24.	WHILE ringStatus.1 AND 0x10 ≠ 0x10 (xxx1xxxx = maxout)	
25.	DELAY 1 Second	
26.	GET ringStatus.2	
27.	WEND *Note:* Wait for Max Out Indication on Ring 2. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
28.	GET phaseStatusGroupPhaseOns.1	
29.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x20= 0x20 *Note:* Verify that Max Out indication occurred on Phase 6.	Pass/Fail
30.	Set HITL Detector Input 8 and 14 = Off	
31.	DELAY .2 Seconds	
Detector 14 extends Phase 13 when Phase 13 is Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	

11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 3 = On	
16.	DELAY .2 Seconds	
17.	Set HITL Detector Input 14 = On	
18.	DELAY .2 Seconds	
19.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
20.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x01 AND phaseStatusGroupPhaseOns.2 ≠ 0x10	
21.	DELAY 1 Second	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WEND *Note:* Wait until controller reaches 1+13.	
24.		
25.	Set HITL Detector Input 3 = Off	
26.	DELAY .2 Seconds	
27.	GET ringStatus.2	
28.	WHILE ringStatus.2 AND 0x10 ≠ 0x10 (xxx1xxxx = maxout)	
29.	DELAY 1 Second	
30.	GET ringStatus.2	
31.	WEND *Note:* Wait for Max Out Indication on Ring 2. Ring 1 = 2, 3, 4, 9, 11, & 12 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
32.	GET phaseStatusGroupPhaseOns.2	
33.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.2 AND 0x10 = 0x10 *Note:* Ensure that Max Out occurred on Phase 13.	Pass/Fail
34.	Set HITL Detector Input 14 = Off	
35.	DELAY .2 Seconds	
Detector 14 Teardown		
	PERFORM Detector Teardown – TC020 if not proceeding to another detector operation test case	
	Test Case Results	
Tested By:		Date Tested
Test Case Notes:		
Version History:	v1.00 04/12/06 Initial Draft – RDR v1.01 07/05/06 Deleted strikethroughs, moved initial detector turnoff point, corrected detector number in some cases, and updated notes – RDR v1.02 07/17/06 Implemented script and proofed – JJ	

Detector 15 Operations

Test Case: **TC015**	*Title*:	**Detector 15 Operations**	
	Description:	Verifies the operation of Detector 15 to call and extends phase 6 under specific conditions.	
	Constants:		
	Variables:		
	Pass/Fail Criteria:	The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*		*Results*
Setup			
	PERFORM Detector Operations Setup – TC019 if not already done so.		
Detector 15 calls Phase 6 during 2+5 Green			
1.	GET ringStatus.1, ringStatus.2		
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03		
3.	DELAY 1 Second		
4.	GET ringStatus.1, ringStatus.2		
5.	WEND *Note:* Loop until controller rests somewhere.		
6.	Set HITL Detector Input 2 = On		
7.	DELAY .2 Seconds		
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2		
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x12 AND phaseStatusGroupPhaseOns.2 ≠ 0X00		
10.	DELAY 1 Second		
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2		
12.	WEND *Note:* Wait until controller reaches 2+5.		
13.	Set HITL Detector Input 2 = Off		
14.	DELAY .2 Seconds		
15.	Set HITL Detector Input 15 = On		
16.	DELAY .2 Seconds		
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1		
18.	WHILE phaseStatusGroupGreens.1 = 0x12 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x20 ≠ 0x20		
19.	DELAY 1 Second		
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1		
21.	WEND		
22.	VERIFY phaseStatusGroupGreens.1 = 0x12 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x20 = 0x20 *Note:* Ensure that Phase 6 registers a Vehicle Call during 2+5 Green.		Pass/Fail
23.	Set HITL Detector Input 15 = Off		
24.	DELAY .2 Seconds		

Detector 15 calls Phase 6 during 3+5 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 15 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x14 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x20 ≠ 0x20	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x14 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x20 = 0x20 *Note:* Ensure that Phase 6 registers a Vehicle Call during 3+5 Green.	Pass/Fail
23.	Set HITL Detector Input 15 = Off	
24.	DELAY .2 Seconds	
Detector 15 calls Phase 6 during 4+5 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	

12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 15 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x18 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x20 ≠ 0x20	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x18 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x20 = 0x20 *Note:* Ensure that Phase 6 registers a Vehicle Call during 4+5 Green.	Pass/Fail
23.	Set HITL Detector Input 15 = Off	
24.	DELAY .2 Seconds	
Detector 15 calls Phase 6 during 5+9 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 7 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x10 AND phaseStatusGroupPhaseOns.2 ≠ 0x01	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 5+9.	
22.	Set HITL Detector Input 7 = Off	
23.	DELAY .2 Seconds	

24.	Set HITL Detector Input 15 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x10 AND phaseStatusGroupGreens.2 = 0x01 AND phaseStatusGroupVehCalls.1 AND 0x20 ≠ 0x20	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x10 AND phaseStatusGroupGreens.2 = 0x01 AND phaseStatusGroupVehCalls.1 AND 0x20 = 0x20 *Note:* Ensure that Phase 6 registers a Vehicle Call during 5+9 Green.	Pass/Fail
32.	Set HITL Detector Input 15 = Off	
33.	DELAY .2 Seconds	
Detector 15 calls Phase 6 during 1+7 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 15 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x41 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x20 ≠ 0x20	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x41 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x20 = 0x20 *Note:* Ensure that Phase 6 registers a Vehicle Call during 1+7 Green.	Pass/Fail

23.	Set HITL Detector Input 15 = Off	
24.	DELAY .2 Seconds	
Detector 15 calls Phase 6 during 1+8 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 15 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x81 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x20 ≠ 0x20	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x81 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x20 = 0x20 *Note:* Ensure that Phase 6 registers a Vehicle Call during 1+8 Green.	Pass/Fail
23.	Set HITL Detector Input 15 = Off	
24.	DELAY .2 Seconds	
Detector 15 calls Phase 6 during 2+16 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	

11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 2 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x02 AND phaseStatusGroupPhaseOns.2 ≠ 0x80	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 2+16.	
22.	Set HITL Detector Input 2 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 15 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x80 AND phaseStatusGroupVehCalls.1 AND 0x20 ≠ 0x20	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x80 AND phaseStatusGroupVehCalls.1 AND 0x20 = 0x20 *Note:* Ensure that Phase 6 registers a Vehicle Call during 2+16 Green.	Pass/Fail
32.	Set HITL Detector Input 15 = Off	
33.	DELAY .2 Seconds	
Detector 15 calls Phase 6 during 2+15 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	

12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 2 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x02 AND phaseStatusGroupPhaseOns.2 ≠ 0x40	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 2+15.	
22.	Set HITL Detector Input 2 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 15 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x40 AND phaseStatusGroupVehCalls.1 AND 0x20 ≠ 0x20	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x40 AND phaseStatusGroupVehCalls.1 AND 0x20 = 0x20 *Note:* Ensure that Phase 6 registers a Vehicle Call during 2+15 Green.	Pass/Fail
32.	Set HITL Detector Input 15 = Off	
33.	DELAY .2 Seconds	
Detector 15 calls Phase 6 during 1+13 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 6 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+6.	
13.	Set HITL Detector Input 6 = Off	

14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 4 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x01 AND phaseStatusGroupPhaseOns.2 ≠ 0x10	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 1+13.	
22.	Set HITL Detector Input 4 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 15 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x01 AND phaseStatusGroupGreens.2 = 0x10 AND phaseStatusGroupVehCalls.1 AND 0x20 ≠ 0x20	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x01 AND phaseStatusGroupGreens.2 = 0x10 AND phaseStatusGroupVehCalls.1 AND 0x20 = 0x20 *Note:* Ensure that Phase 6 registers a Vehicle Call during 1+13 Green.	Pass/Fail
32.	Set HITL Detector Input 15 = Off	
33.	DELAY .2 Seconds	
Detector 15 extends Phase 6 when Phase 6 is Green until a gap in Detector 15 activity occurs at which time Detector 15 becomes inactive until Phase 6 Yellow		
1.	GET vehicleDetectorExtend.15 and RECORD RESPONSE VALUE in [currentExtendValue]	
2.	SET vehicleDetectorExtend.15 = 40 *Note:* Set Detector 15 extend time = 4 seconds.	
3.	GET ringStatus.1, ringStatus.2	
4.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
5.	DELAY 1 Second	
6.	GET ringStatus.1, ringStatus.2	
7.	WEND *Note:* Loop until controller rests somewhere.	
8.	Set HITL Detector Input 2 = On	
9.	DELAY .2 Seconds	
10.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
11.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x12 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
12.	DELAY 1 Second	

13.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
14.	WEND *Note:* Wait until controller reaches 2+5.	
15.	Set HITL Detector Input 2= Off	
16.	DELAY .2 Seconds	
17.	Set HITL Detector Input 15 = On (To call and then extend Phase 6)	
18.	DELAY .2 Seconds	
19.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
20.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
21.	DELAY 1 Second	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WEND *Note:* Wait until controller reaches 1+6.	
24.	Set HITL Detector Input 3 = On (So that Detector 15 will extend Phase 6 when there is an opposing call)	
25.	DELAY .2 Seconds	
26.	GET ringStatus.2	
27.	WHILE ringStatus.2 AND 0x07 ≠ 0x01 (xxxxx001 = extension)	
28.	DELAY .1 Second	
29.	GET ringStatus.2	
30.	WEND *Note:* Wait for indication that extensions are timing. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, ***6***, 7, 8, & 13	
31.	GET phaseStatusGroupPhaseOns.1	
32.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x20 = 0x20 *Note:* Verify that the extensions are on Phase 6.	Pass/Fail
33.	Set HITL Detector Input 15 = Off	
34.	DELAY 3 Seconds	
35.	Set HITL Detector Input 15 = On	
36.	DELAY .2 Seconds	
37.	Set HITL Detector Input 15 = Off	
38.	DELAY 2.8 Seconds	
39.	GET ringStatus.2	
40.	VERIFY that RESPONSE VALUE ringStatus.2 AND 0x07 = 0x01 (xxxxx001 = extension) *Note:* Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, ***6***, 7, 8, & 13	
41.	GET phaseStatusGroupPhaseOns.1	
42.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x20 = 0x20 *Note:* Verify that extensions are still timing on Phase 6 because call was entered < 4 seconds later.	Pass/Fail
43.	Set HITL Detector Input 15 = On	
44.	DELAY .2 Seconds	
45.	Set HITL Detector Input 15 = Off	

46.	DELAY 4.8 Seconds *Note:* Since the time between actuations is now 5 seconds, the timer will gap and therefore disable Detector 15 from putting in any further extensions.	
47.	Set HITL Detector Input 15 = On	
48.	DELAY .2 Seconds	
49.	GET ringStatus.2	
50.	VERIFY that RESPONSE VALUE ringStatus.2 AND 0x07 = 0x04 (xxxxx100 = Yellow Change) *Note:* Verify that the Yellow Change is on Phase 6 because call was entered > 4 seconds later. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, ***6***, 7, 8, & 13	
51.	GET phaseStatusGroupPhaseOns.1, ringStatus.2, phaseStatusGroupVehCalls.1	
52.	WHILE (phaseStatusGroupPhaseOns.1 AND 0x20 = 0x20) AND (ringStatus.2 AND 0x07 = 0x04) AND (phaseStatusGroupVehCalls.1 AND 0x20 ≠ 0x20)	
53.	DELAY 1 Second	
54.	GET phaseStatusGroupPhaseOns.1, ringStatus.2, phaseStatusGroupVehCalls.1	
55.	WEND	
56.	VERIFY (phaseStatusGroupPhaseOns.1 AND 0x20 = 0x20) AND (ringStatus.2 AND 0x07 = 0x04) AND (phaseStatusGroupVehCalls.1 AND 0x20 = 0x20) *Note:* Ensure that Phase 6 registers a Vehicle Call during 6 Yellow.	Pass/Fail
57.	Set HITL Detector Input 3 = Off and 15 = Off	
58.	DELAY .2 Seconds	
59.	SET vehicleDetectorExtend.15 = [currentExtendValue], *Note:* Restore original values.	
Teardown		
	PERFORM Detector Teardown – TC020 if not proceeding to another detector operation test case	

		Test Case Results		
Tested By:		Date Tested		Pass/Fail
Test Case Notes:	<notes>			
Version History:	v1.00 05/05/06 Initial Draft – RDR v1.01 07/05/06 Updated notes – RDR v1.02 07/12/06 Implemented script and proofed – JJ			

Detector 16 Operations

Test Case: **TC016**	*Title*:	**Detector 16 Operations**	
	Description:	Verifies the operation of Detector 16 to call and extend Phase 8 under specific conditions and extend interval 1825B (2+16).	
	Constants:		
	Variables:	currentExtendValue	
	Pass/Fail Criteria:	The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*		*Results*
Setup			
	PERFORM Detector Operations Setup – TC019 if not already done so.		
Detector 16 calls Phase 8 when Phase 8 not Green (In 2+16 Green)			
1.	GET ringStatus.1, ringStatus.2		
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03		
3.	DELAY 1 Second		
4.	GET ringStatus.1, ringStatus.2		
5.	WEND *Note:* Loop until controller rests somewhere.		
6.	Set HITL Detector Input 8 = On		
7.	DELAY .2 Seconds		
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2		
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00		
10.	DELAY 1 Second		
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2		
12.	WEND *Note:* Wait until controller reaches 1+8.		
13.	Set HITL Detector Input 8 = Off		
14.	DELAY .2 Seconds		
15.	Set HITL Detector Input 2 = On		
16.	DELAY .2 Seconds		
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2		
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x02 AND phaseStatusGroupPhaseOns.2 ≠ 0x80		
19.	DELAY 1 Second		
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2		
21.	WEND *Note:* Wait until controller reaches 2+16.		
22.	Set HITL Detector Input 2 = Off		
23.	DELAY .2 Seconds		
24.	Set HITL Detector Input 16 = On		
25.	DELAY .2 Seconds		
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1		

27.	WHILE phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x80 AND phaseStatusGroupVehCalls.1 AND 0x80 ≠ 0x080	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x80 AND phaseStatusGroupVehCalls.1 AND 0x80 = 0x80 *Note:* Ensure that Phase 8 registers a Vehicle Call during 2+16 Green.	Pass/Fail
32.	Set HITL Detector Input 16 = Off	
33.	DELAY .2 Seconds	
Detector 16 calls Phase 8 when Phase 8 not Green (In 2+15 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 2 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x02 AND phaseStatusGroupPhaseOns.2 ≠ 0x40	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 2+15.	
22.	Set HITL Detector Input 2 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 16 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	

27.	WHILE phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x40 AND phaseStatusGroupVehCalls.1 AND 0x80 ≠ 0x080	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x40 AND phaseStatusGroupVehCalls.1 AND 0x80 = 0x80 *Note:* Ensure that Phase 8 registers a Vehicle Call during 2+15 Green.	Pass/Fail
32.	Set HITL Detector Input 16 = Off	
33.	DELAY .2 Seconds	
Detector 16 calls Phase 8 when Phase 8 not Green (In 1+13 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 3 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x01 AND phaseStatusGroupPhaseOns.2 ≠ 0x10	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 1+13.	
22.	Set HITL Detector Input 3 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 16 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	

27.	WHILE phaseStatusGroupGreens.1 = 0x01 AND phaseStatusGroupGreens.2 = 0x10 AND phaseStatusGroupVehCalls.1 AND 0x80 ≠ 0x080	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x01 AND phaseStatusGroupGreens.2 = 0x10 AND phaseStatusGroupVehCalls.1 AND 0x80 = 0x80 *Note:* Ensure that Phase 8 registers a Vehicle Call during 1+13 Green.	Pass/Fail
32.	Set HITL Detector Input 16 = Off	
33.	DELAY .2 Seconds	
Detector 16 calls Phase 8 when Phase 8 not Green (In 2+5 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 2 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x12 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 2+5.	
13.	Set HITL Detector Input 2 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 16 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x12 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x80 ≠ 0x80	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x12 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x80 = 0x80 *Note*: Ensure that Phase 8 registers a Vehicle Call during 2+5 Green.	Pass/Fail
23.	Set HITL Detector Input 16 = Off	
24.	DELAY .2 Seconds	

Detector 16 calls Phase 8 when Phase 8 not Green (In 3+5 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 16 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x14 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x80 ≠ 0x80	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x14 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x80 = 0x80 *Note*: Ensure that Phase 8 registers a Vehicle Call during 3+5 Green.	Pass/Fail
23.	Set HITL Detector Input 16 = Off	
24.	DELAY .2 Seconds	
Detector 16 calls Phase 8 when Phase 8 not Green (In 4+5 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	

11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 16 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x18 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x80 ≠ 0x80	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x18 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x80 = 0x80 *Note*: Ensure that Phase 8 registers a Vehicle Call during 4+5 Green.	Pass/Fail
23.	Set HITL Detector Input 16 = Off	
24.	DELAY .2 Seconds	
Detector 16 calls Phase 8 when Phase 8 not Green (In 6+12 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 6 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x20 AND phaseStatusGroupPhaseOns.2 ≠ 0x08	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	

21.	WEND *Note:* Wait until controller reaches 6+12.	
22.	Set HITL Detector Input 6 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 16 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x08 AND phaseStatusGroupVehCalls.1 AND 0x80 ≠ 0x80	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x08 AND phaseStatusGroupVehCalls.1 AND 0x80 = 0x80 *Note*: Ensure that Phase 8 registers a Vehicle Call during 6+12 Green.	Pass/Fail
32.	Set HITL Detector Input 16 = Off	
33.	DELAY .2 Seconds	
Detector 16 calls Phase 8 when Phase 8 not Green (In 6+11 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 6 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x20 AND phaseStatusGroupPhaseOns.2 ≠ 0x04	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	

21.	WEND *Note:* Wait until controller reaches 6+11.	
22.	Set HITL Detector Input 6 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 16 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x04 AND phaseStatusGroupVehCalls.1 AND 0x80 ≠ 0x80	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x04 AND phaseStatusGroupVehCalls.1 AND 0x80 = 0x80 *Note*: Ensure that Phase 8 registers a Vehicle Call during 6+11 Green.	Pass/Fail
32.	Set HITL Detector Input 16 = Off	
33.	DELAY .2 Seconds	
Detector 16 calls Phase 8 when Phase 8 not Green (In 5+9 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 7 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x10 AND phaseStatusGroupPhaseOns.2 ≠ 0x01	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	

21.	WEND *Note:* Wait until controller reaches 5+9.	
22.	Set HITL Detector Input 7 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 16 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x10 AND phaseStatusGroupGreens.2 = 0x01 AND phaseStatusGroupVehCalls.1 AND 0x80 ≠ 0x80	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x10 AND phaseStatusGroupGreens.2 = 0x01 AND phaseStatusGroupVehCalls.1 AND 0x80 = 0x80 *Note*: Ensure that Phase 8 registers a Vehicle Call during 5+9 Green.	Pass/Fail
32.	Set HITL Detector Input 16 = Off	
33.	DELAY .2 Seconds	
Detector 16 calls Phase 8 when Phase 8 not Green (In 1+6 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 6 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+6.	
13.	Set HITL Detector Input 6 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 16 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x21 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x80 ≠ 0x80	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	

21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x21 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x80= 0x80 *Note*: Ensure that Phase 8 registers a Vehicle Call during 1+6 Green.	Pass/Fail
23.	Set HITL Detector Input 16 = On	
24.	DELAY .2 Seconds	
Detector 16 calls Phase 8 when Phase 8 not Green (In 1+7 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 16 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x41 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x80 ≠ 0x80	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x41 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x80 = 0x80 *Note*: Ensure that Phase 8 registers a Vehicle Call during 1+7 Green.	Pass/Fail
23.	Set HITL Detector Input 16 = Off	
24.	DELAY .2 Seconds	
Detector 16 extends Phase 8 until Phase 2 call exists and Detector 16 gaps and Phase 2 call continues		
1.	GET vehicleDetectorExtend.16 = [currentExtendValue]	
2.	SET vehicleDetectorExtend.16 = 40 *Note:* Set Detector 16 extend time = 4 seconds so that actuation less than 4 seconds apart keep the phase extending.	

3.	GET ringStatus.1, ringStatus.2	
4.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
5.	DELAY 1 Second	
6.	GET ringStatus.1, ringStatus.2	
7.	WEND *Note:* Loop until controller rests somewhere.	
8.	Set HITL Detector Input 4 = On	
9.	DELAY .2 Seconds	
10.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
11.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
12.	DELAY 1 Second	
13.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
14.	WEND *Note:* Wait until controller reaches 4+5 first so that Detector 16 can call and extend Phase 8.	
15.	Set HITL Detector Input 4 = Off	
16.	DELAY .2 Seconds	
17.	Set HITL Detector Input 16 = On (To call and then extend Phase 8)	
18.	DELAY .2 Seconds	
19.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
20.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
21.	DELAY 1 Second	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WEND *Note:* Wait until controller reaches 1+8.	
24.	Set HITL Detector Input 2 = On (Detector 16 will extend Phase 8 when there is an opposing call)	
25.	DELAY .2 Seconds *Note:* This should get Phase 4 to start timing extensions and sets up conditional logic.	
26.	GET ringStatus.2	
27.	WHILE ringStatus.2 AND 0x07 ≠ 0x01 (xxxxx001 = extension)	
28.	DELAY .1 Second	
29.	GET ringStatus.2	
30.	WEND *Note:* Wait for indication that extensions are timing. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
31.	GET phaseStatusGroupPhaseOns.1	
32.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x80 = 0x80 *Note:* Verify that the extensions are on Phase 8.	Pass/Fail
33.	Set HITL Detector Input 16 = Off	
34.	DELAY 3 Seconds	

35.	Set HITL Detector Input 16 = On	
36.	DELAY .2 Seconds	
37.	Set HITL Detector Input 16 = Off	
38.	DELAY 2.8 Seconds	
39.	GET ringStatus.2	
40.	VERIFY that RESPONSE VALUE ringStatus.2 AND 0x07 = 0x01 (xxxxx001 = extension) *Note:* Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	Pass/Fail
41.	GET phaseStatusGroupPhaseOns.1	
42.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x80 = 0x80 *Note:* Verify that extensions are still timing on Phase 8 because call was entered < 4 seconds later.	
43.	Set HITL Detector Input 16 = On	
44.	DELAY .2 Seconds	
45.	Set HITL Detector Input 16 = Off	
46.	DELAY 2.8 Seconds	
47.	GET ringStatus.2	
48.	VERIFY that RESPONSE VALUE ringStatus.2 AND 0x07 = 0x01 (xxxxx001 = extension) *Note:* Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	Pass/Fail
49.	GET phaseStatusGroupPhaseOns.1	
50.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x80 = 0x80 *Note:* Verify that extensions are still timing on Phase 8 because call was entered < 4 seconds later.	Pass/Fail
51.	Set HITL Detector Input 16 = On	
52.	DELAY .2 Seconds	
53.	Set HITL Detector Input 16 = Off	
54.	DELAY 4.8 Seconds *Note:* Since the time between actuations is now 5 seconds, the timer will gap and therefore disable Detector 16 from putting in any further extensions.	
55.	Set HITL Detector Input 16 = On	
56.	DELAY .2 Seconds	
57.	GET ringStatus.2	
58.	VERIFY that RESPONSE VALUE ringStatus.2 AND 0x07 ≠ 0x01 (xxxxx001 = extension) *Note:* Verify that Phase 8 is no longer timing extensions. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	Pass/Fail
59.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupVehCalls.1	
60.	WHILE (phaseStatusGroupPhaseOns.1 AND 0x80 = 0x80) AND (phaseStatusGroupVehCalls.1 AND 0x80 ≠ 0x80)	
61.	DELAY 1 Second	
62.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupVehCalls.1	

63.	WEND	
64.	VERIFY (phaseStatusGroupPhaseOns.1 AND 0x80 = 0x80) AND (phaseStatusGroupVehCalls.1 AND 0x80 = 0x80) *Note:* Ensure that Phase 8 registers a Vehicle Call.	Pass/Fail
65.	Set HITL Detector Input 2 = Off and 16 = Off	
66.	DELAY .2 Seconds	
	POST-CONDITION The Detector 16 vehicleDetectorExtend is still set to 4 seconds	
Detector 16 extends Phase 8 until Phase 2 call exists and Detector 16 gaps but not if Phase 2 call disappears		
	PRE-CONDITION The Detector 16 vehicleDetectorExtend is still set to 4 seconds	
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 16 = On (To call and then extend 1+8)	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 1+8.	
22.	Set HITL Detector Input 2 = On	
23.	DELAY .2 Seconds *Note:* Detector 12 will extend Phase 4 when there is an opposing call.	
24.	GET ringStatus.2	
25.	WHILE ringStatus.2 AND 0x07 ≠ 0x01 (xxxxx001 = extension)	
26.	DELAY .1 Second	
27.	GET ringStatus.2	

28.	WEND *Note:* Wait for indication that extensions are timing. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
29.	GET phaseStatusGroupPhaseOns.1	
30.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x80 = 0x80 *Note:* Verify that the extensions are timing on Phase 8.	Pass/Fail
31.	Set HITL Detector Input 16 = Off	
32.	DELAY 3 Seconds	
33.	Set HITL Detector Input 16 = On	
34.	DELAY .2 Seconds	
35.	Set HITL Detector Input 16 = Off	
36.	DELAY 2.8 Seconds	
37.	GET ringStatus.2	
38.	VERIFY that RESPONSE VALUE ringStatus.2 AND 0x07 = 0x01 (xxxxx001 = extension) Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	Pass/Fail
39.	GET phaseStatusGroupPhaseOns.1	
40.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x80 = 0x80 *Note:* Verify that extensions are still timing on Phase 8.	Pass/Fail
41.	Set HITL Detector Input 16 = On	
42.	DELAY .2 Seconds	
43.	Set HITL Detector Input 16 = Off	
44.	DELAY 2.8 Seconds	
45.	GET ringStatus.2	
46.	VERIFY that RESPONSE VALUE ringStatus.2 AND 0x07 = 0x01 (xxxxx001 = extension) *Note:* Wait for indication that extensions are timing. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	Pass/Fail
47.	GET phaseStatusGroupPhaseOns.1	
48.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x80 = 0x80 *Note:* Verify that extensions are still timing on Phase 8.	Pass/Fail
49.	Set HITL Detector Input 16 = On	
50.	DELAY .2 Seconds	
51.	Set HITL Detector Input 16 = Off, 8 = On, 2 = Off and 3 = On *Note:* Stop extending Phase 8 via Detector 16, turn on Detector 8 to keep Phase 8 extending, turn off Detector 2 opposing demand as part of the logic, and turn on Detector 3 to place opposing demand Detector 8 will keep extending Phase 8 irrespective of Detector 16, no call on Phase 2 will reset Detector 16 gap function, and the call on Phase 3 will enable extensions to continue to time.	

52.	DELAY 4.8 Seconds *Note:* This would have the effect of allowing the extend timer to gap and therefore disable detector 16 but since Phase 2 no longer has a call, another activation of Detector 16 will continue to extend Phase 8.	
53.	Set HITL Detector Input 16 = On and 8 Off	
54.	DELAY .2 Seconds	
55.	Set HITL Detector Input 16 = Off	
56.	DELAY .2 Seconds	
57.	GET ringStatus.2	
58.	VERIFY that RESPONSE VALUE ringStatus.2 AND 0x07 = 0x01 (xxxxx001 = extension) *Note:* Wait for indication that extensions are timing. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	Pass/Fail
59.	GET phaseStatusGroupPhaseOns.1	
60.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x80 = 0x80 *Note:* Verify that extensions are still timing on Phase 8 because even though call was > 4 seconds later, the absence of call on Phase 2 allowed Phase 8 to continue to continue to time extensions.	Pass/Fail
61.	Set HITL Detector Input 3 = Off	
62.	DELAY .2 Seconds	
63.	SET vehicleDetectorExtend.16 = [currentExtendValue] *Note:* Return Detector 16 extend time back to their original value.	
Detector 16 extends Phase 16 when Phase 16 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 16 = On	
16.	DELAY .2 Seconds	
17.	Set HITL Detector Input 2 = On	
18.	DELAY .2 Seconds	

19.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
20.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x02 AND phaseStatusGroupPhaseOns.2 ≠ 0x80	
21.	DELAY 1 Second	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WEND *Note:* Wait until controller reaches 2+16.	
24.	Set HITL Detector Input 2 = Off	
25.	DELAY .2 Seconds	
26.	GET ringStatus.2	
27.	WHILE ringStatus.2 AND 0x10 ≠ 0x10 (xxx1xxxx = maxout)	
28.	DELAY 1 Second	
29.	GET ringStatus.2	
30.	WEND *Note:* Wait for Max Out Indication. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
31.	GET phaseStatusGroupPhaseOns.2	
32.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.2 AND 0x80 = 0x80 *Note:* Ensure that Max Out occurred on Phase 16.	Pass/Fail
33.	Set HITL Detector Input 16 = Off	
34.	DELAY .2 Seconds	
Teardown		
	PERFORM Detector Teardown – TC020 if not proceeding to another detector operation test case	
Test Case Results		
Tested By:	Date Tested	
Test Case Notes:		
Version History:	v1.00 05/09/06 Initial Draft – RDR v1.01 07/05/06 Updated notes – RDR v1.02 07/12/06 Implemented script and proofed – JJ	

Detector 17 Operations

Test Case: **TC017**	*Title*: **Detector 17 Operations** Description: Verifies the operation of Detector 17 to call and extend Phase 3 under specific conditions and extend interval 3516B (6+11). Constants: Variables: currentExtendValue Pass/Fail Criteria: The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*	*Results*
Setup		
	PERFORM Detector Operations Setup – TC019 if not already done so.	
Detector 17 calls Phase 3 when Phase 3 not Green (In 2+16 Green)		

1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x0	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 2 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x20 AND phaseStatusGroupPhaseOns.2 ≠ 0x80	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 2+16.	
22.	Set HITL Detector Input 2 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 17 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x80 AND phaseStatusGroupVehCalls.1 AND 0x04 ≠ 0x04	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x80 AND phaseStatusGroupVehCalls.1 AND 0x04 = 0x04 *Note:* Ensure that Phase 3 registers a Vehicle Call during 2+16 Green.	Pass/Fail
32.	Set HITL Detector Input 17 = Off	
33.	DELAY .2 Seconds	
Detector 17 calls Phase 3 when Phase 3 not Green (In 2+15 Green)		
1.	GET ringStatus.1, ringStatus.2	

2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 2 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x02 AND phaseStatusGroupPhaseOns.2 ≠ 0x40	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 2+15.	
22.	Set HITL Detector Input 2 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 17 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x04 AND phaseStatusGroupVehCalls.1 AND 0x04 ≠ 0x04	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x40 AND phaseStatusGroupVehCalls.1 AND 0x04 = 0x04 *Note:* Ensure that Phase 3 registers a Vehicle Call during 2+15 Green.	Pass/Fail
32.	Set HITL Detector Input 17 = Off	
33.	DELAY .2 Seconds	
Detector 17 calls Phase 3 when Phase 3 not Green (In 1+13 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	

3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 3 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x01 AND phaseStatusGroupPhaseOns.2 ≠ 0x10	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 1+13.	
22.	Set HITL Detector Input 3 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 17 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x01 AND phaseStatusGroupGreens.2 = 0x10 AND phaseStatusGroupVehCalls.1 AND 0x04 ≠ 0x04	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x01 AND phaseStatusGroupGreens.2 = 0x10 AND phaseStatusGroupVehCalls.1 AND 0x04 = 0x04 *Note:* Ensure that Phase 3 registers a Vehicle Call during 1+13 Green.	Pass/Fail
32.	Set HITL Detector Input 17 = Off	
33.	DELAY .2 Seconds	
Detector 17 calls Phase 3 when Phase 3 not Green (In 2+5 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	

4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 2 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x12 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 2+5.	
13.	Set HITL Detector Input 2 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 17 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x12 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x04 ≠ 0x04	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x12 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x04 = 0x04 *Note*: Ensure that Phase 3 registers a Vehicle Call during 2+5 Green.	Pass/Fail
23.	Set HITL Detector Input 17 = Off	
24.	DELAY .2 Seconds	
Detector 17 calls Phase 3 when Phase 3 not Green (In 4+5 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	

13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 17 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x18 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x04 ≠ 0x04	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x18 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x04 = 0x04 *Note*: Ensure that Phase 3 registers a Vehicle Call during 4+5 Green.	Pass/Fail
23.	Set HITL Detector Input 17 = Off	
24.	DELAY .2 Seconds	
Detector 17 calls Phase 3 when Phase 3 not Green (In 6+12 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 6 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x20 AND phaseStatusGroupPhaseOns.2 ≠ 0x08	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 6+12.	
22.	Set HITL Detector Input 6 = Off	
23.	DELAY .2 Seconds	

24.	Set HITL Detector Input 17 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x08 AND phaseStatusGroupVehCalls.1 AND 0x04 ≠ 0x04	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x08 AND phaseStatusGroupVehCalls.1 AND 0x04 = 0x04 *Note*: Ensure that Phase 3 registers a Vehicle Call during 6+12 Green.	Pass/Fail
32.	Set HITL Detector Input 17 = Off	
33.	DELAY .2 Seconds	
Detector 17 calls Phase 8 when Phase 8 not Green (In 6+11 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 6 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x20 AND phaseStatusGroupPhaseOns.2 ≠ 0x04	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 6+11.	
22.	Set HITL Detector Input 6 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 17 = On	

25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x04 AND phaseStatusGroupVehCalls.1 AND 0x04 ≠ 0x04	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x04 AND phaseStatusGroupVehCalls.1 AND 0x04 = 0x04 *Note*: Ensure that Phase 3 registers a Vehicle Call during 6+11 Green.	Pass/Fail
32.	Set HITL Detector Input 17 = Off	
33.	DELAY .2 Seconds	
Detector 17 calls Phase 3 when Phase 3 not Green (In 5+9 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 7 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x10 AND phaseStatusGroupPhaseOns.2 ≠ 0x01	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 5+9.	
22.	Set HITL Detector Input 7 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 17 = On	
25.	DELAY .2 Seconds	

26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x10 AND phaseStatusGroupGreens.2 = 0x01 AND phaseStatusGroupVehCalls.1 AND 0x04 ≠ 0x04	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x10 AND phaseStatusGroupGreens.2 = 0x01 AND phaseStatusGroupVehCalls.1 AND 0x04 = 0x04 *Note*: Ensure that Phase 3 registers a Vehicle Call during 5+9 Green.	Pass/Fail
32.	Set HITL Detector Input 17 = Off	
33.	DELAY .2 Seconds	
Detector 17 calls Phase 3 when Phase 3 not Green (In 1+6 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 6 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+6.	
13.	Set HITL Detector Input 6 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 17 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x21 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x04 ≠ 0x04	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x21 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x04= 0x04 *Note*: Ensure that Phase 3 registers a Vehicle Call during 1+6 Green.	Pass/Fail
23.	Set HITL Detector Input 17 = On	

24.	DELAY .2 Seconds	
Detector 17 calls Phase 3 when Phase 3 not Green (In 1+7 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 17 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x41 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x04 ≠ 0x04	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x41 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x04 = 0x04 *Note*: Ensure that Phase 3 registers a Vehicle Call during 1+7 Green.	Pass/Fail
23.	Set HITL Detector Input 17 = Off	
24.	DELAY .2 Seconds	
Detector 17 calls Phase 3 when Phase 3 not Green (In 1+8 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	

11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 17 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x81 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x04 ≠ 0x04	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x81 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x04 = 0x04 *Note*: Ensure that Phase 3 registers a Vehicle Call during 1+8 Green.	Pass/Fail
23.	Set HITL Detector Input 17 = Off	
24.	DELAY .2 Seconds	
Detector 17 extends Phase 3 until Phase 6 call exists and Detector 17 gaps and Phase 6 call continues		
1.	GET vehicleDetectorExtend.17 = [currentExtendValue]	
2.	SET vehicleDetectorExtend.17 = 40 *Note:* Set Detector 17 extend time = 4 seconds so that actuations < 4 seconds apart keep the phase extending.	
3.	GET ringStatus.1, ringStatus.2	
4.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
5.	DELAY 1 Second	
6.	GET ringStatus.1, ringStatus.2	
7.	WEND *Note:* Loop until controller rests somewhere.	
8.	Set HITL Detector Input 8 = On	
9.	DELAY .2 Seconds	
10.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
11.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
12.	DELAY 1 Second	
13.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
14.	WEND *Note:* Wait until controller reaches 1+8 first so that Detector 17 can call and extend Phase 3.	
15.	Set HITL Detector Input 8 = Off	
16.	DELAY .2 Seconds	
17.	Set HITL Detector Input 17 = On (To call and then extend Phase 3)	

18.	DELAY .2 Seconds	
19.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
20.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
21.	DELAY 1 Second	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WEND *Note:* Wait until controller reaches 3+5.	
24.	Set HITL Detector Input 6 = On (Detector 17 will extend Phase 3 when there is an opposing call)	
25.	DELAY .2 Seconds *Note:* This should get Phase 3 to start timing extensions and sets up conditional logic.	
26.	GET ringStatus.1	
27.	WHILE ringStatus.1 AND 0x07 ≠ 0x01 (xxxxx001 = extension)	
28.	DELAY .1 Second	
29.	GET ringStatus.1	
30.	WEND *Note:* Wait for indication that extensions are timing. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
31.	GET phaseStatusGroupPhaseOns.1	
32.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x04 = 0x04 *Note:* Verify that the extensions are on Phase 3.	Pass/Fail
33.	Set HITL Detector Input 17 = Off	
34.	DELAY 3 Seconds	
35.	Set HITL Detector Input 17 = On	
36.	DELAY .2 Seconds	
37.	Set HITL Detector Input 17 = Off	
38.	DELAY 2.8 Seconds	
39.	GET ringStatus.1	
40.	VERIFY that RESPONSE VALUE ringStatus.1 AND 0x07 = 0x01 (xxxxx001 = extension) *Note:* Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	Pass/Fail
41.	GET phaseStatusGroupPhaseOns.1	
42.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x04 = 0x04 *Note:* Verify that extensions are still timing on Phase 3 because call was entered < 4 seconds later.	Pass/Fail
43.	Set HITL Detector Input 17 = On	
44.	DELAY .2 Seconds	
45.	Set HITL Detector Input 17 = Off	
46.	DELAY 2.8 Seconds	
47.	GET ringStatus.1	

48.	VERIFY that RESPONSE VALUE ringStatus.1 AND 0x07 = 0x01 (xxxxx001 = extension) *Note:* Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	Pass/Fail
49.	GET phaseStatusGroupPhaseOns.1	
50.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x04 = 0x04 *Note:* Verify that extensions are still timing on Phase 3 because call was entered < 4 seconds later.	Pass/Fail
51.	Set HITL Detector Input 17 = On	
52.	DELAY .2 Seconds	
53.	Set HITL Detector Input 17 = Off	
54.	DELAY 4.8 Seconds *Note:* Since the time between actuations is now 5 seconds, the timer will gap and therefore disable Detector 17 from putting in any further extensions.	
55.	Set HITL Detector Input 17 = On	
56.	DELAY .2 Seconds	
57.	GET ringStatus.1	
58.	VERIFY that RESPONSE VALUE ringStatus.1 AND 0x07 ≠ 0x01 (xxxxx001 = extension) *Note:* Verify that Phase 3 is no longer timing extensions. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	Pass/Fail
59.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupVehCalls.1	
60.	WHILE (phaseStatusGroupPhaseOns.1 AND 0x04 = 0x04) AND (phaseStatusGroupVehCalls.1 AND 0x04 ≠ 0x04)	
61.	DELAY 1 Second	
62.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupVehCalls.1	
63.	WEND	
64.	VERIFY (phaseStatusGroupPhaseOns.1 AND 0x04 = 0x04) AND (phaseStatusGroupVehCalls.1 AND 0x04 = 0x04) *Note:* Ensure that Phase 3 registers a Vehicle Call.	Pass/Fail
65.	Set HITL Detector Input 6 = Off and 17 = Off	
66.	DELAY .2 Seconds	
	POST-CONDITION The Detector 17 vehicleDetectorExtend is still set to 4 seconds	
Detector 17 extends Phase 3 until Phase 6 call exists and Detector 17 gaps but not if Phase 6 call disappears		
	PRE-CONDITION The Detector 17 vehicleDetectorExtend is still set to 4 seconds	
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 8 = On	

7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 17 = On (To call and then extend Phase 3)	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 3+5.	
22.	Set HITL Detector Input 6 = On	
23.	DELAY .2 Seconds *Note:* The Detector 17 call gets Phase 3 to start timing extensions because of the opposing demand and sets up conditional logic.	
24.	GET ringStatus.1	
25.	WHILE ringStatus.1 AND 0x07 ≠ 0x01 (xxxxx001 = extension)	
26.	DELAY .1 Second	
27.	GET ringStatus.1	
28.	WEND *Note:* Wait for indication that extensions are timing. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
29.	GET phaseStatusGroupPhaseOns.1	
30.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x04 = 0x04 *Note:* Verify that the extensions are timing on Phase 3.	Pass/Fail
31.	Set HITL Detector Input 17 = Off	
32.	DELAY 3 Seconds	
33.	Set HITL Detector Input 17 = On	
34.	DELAY .2 Seconds	
35.	Set HITL Detector Input 17 = Off	
36.	DELAY 2.8 Seconds	
37.	GET ringStatus.1	

38.	VERIFY that RESPONSE VALUE ringStatus.1 AND 0x07 = 0x01 (xxxxx001 = extension) *Note:* Wait for indication that extensions are timing. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	Pass/Fail
39.	GET phaseStatusGroupPhaseOns.1	
40.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x04 = 0x04 *Note:* Verify that extensions are still timing is on Phase 3.	Pass/Fail
41.	Set HITL Detector Input 17 = On	
42.	DELAY .2 Seconds	
43.	Set HITL Detector Input 17 = Off	
44.	DELAY 2.8 Seconds	
45.	GET ringStatus.1	
46.	VERIFY that RESPONSE VALUE ringStatus.1 AND 0x07 = 0x01 (xxxxx001 = extension) *Note:* Wait for indication that extensions are timing. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	Pass/Fail
47.	GET phaseStatusGroupPhaseOns.1	
48.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x04 = 0x04 *Note:* Verify that extensions are still timing on Phase 3.	Pass/Fail
49.	Set HITL Detector Input 17 = On	
50.	DELAY .2 Seconds	
51.	Set HITL Detector Input 17 = Off, 3 = On, 6 = Off and 7 = On *Note:* Stop extending Phase 3 via Detector 17, turn on Detector 3 to keep Phase 3 extending, turn off Detector 6 opposing demand as part of the logic, and turn on Detector 7 to place opposing demand. Detector 3 will keep extending Phase 3 irrespective of Detector 17, no call on Phase 6 will reset Detector 17 gap function, and the call on Phase 3 will enable extensions to continue to time.	
52.	DELAY 4.8 Seconds *Note:* This would have the effect of allowing the extend timer to gap and therefore disable Detector 17, but since Phase 6 no longer has a call, another activation of Detector 17 will continue to extend Phase 3.	
53.	Set HITL Detector Input 17 = On and 3 Off	
54.	DELAY .2 Seconds	
55.	Set HITL Detector Input 17 = Off	
56.	DELAY .2 Seconds	
57.	GET ringStatus.1	
58.	VERIFY that RESPONSE VALUE ringStatus.1 AND 0x07 = 0x01 (xxxxx001 = extension) *Note:* Wait for indication that extensions are timing. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	Pass/Fail
59.	GET phaseStatusGroupPhaseOns.1	

60.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x04 = 0x04 *Note:* Verify that extensions are still timing on Phase 3 because even though call was > 4 seconds later, the absence of call on Phase 6 allowed Phase 3 to continue to time extensions.	Pass/Fail
61.	Set HITL Detector Input 7 = Off	
62.	DELAY .2 Seconds	
63.	SET vehicleDetectorExtend.17 = [currentExtendValue] *Note:* Return Detector 17 extend time back to its original value.	
Detector 17 extends Phase 11 when Phase 11 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 17 = On	
16.	DELAY .2 Seconds	
17.	Set HITL Detector Input 6 = On	
18.	DELAY .2 Seconds	
19.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
20.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x20 AND phaseStatusGroupPhaseOns.2 ≠ 0x04	
21.	DELAY 1 Second	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WEND *Note:* Wait until controller reaches 6+11.	
24.	Set HITL Detector Input 6 = Off	
25.	DELAY .2 Seconds	
26.	GET ringStatus.1	
27.	WHILE ringStatus.1 AND 0x10 ≠ 0x10 (xxx1xxxx = maxout)	
28.	DELAY 1 Second	
29.	GET ringStatus.1	

30.	WEND *Note:* Wait for Max Out Indication. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
31.	GET phaseStatusGroupPhaseOns.2	
32.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.2 AND 0x04 = 0x04 *Note:* Ensure that Max Out occurred on Phase 11.	Pass/Fail
33.	Set HITL Detector Input 17 = Off	
34.	DELAY .2 Seconds	
Teardown		
	PERFORM Detector Teardown – TC020 if not proceeding to another detector operation test case	
Test Case Results		
Tested By:	Date Tested	
Test Case Notes:		
Version History:	v1.00 05/09/06 Initial Draft – RDR v1.01 07/05/06 Updated notes – RDR v1.02 08/01/06 Implemented script and proofed – JJ	

Detector 18 Operations

Test Case: **TC018**	*Title*:	**Detector 18 Operations**
	Description:	Verifies the operation of Detector 18 to call and extend Phase 7 under specific conditions and extend interval 1725B (2+15).
	Constants:	
	Variables:	currentExtendValue
	Pass/Fail Criteria:	The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.

Test Step Number	*Test Procedure*	*Results*
Setup		
	PERFORM Detector Operations Setup – TC019 if not already done so.	
Detector 18 calls Phase 7 when Phase 7 not Green (In 1+8 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	

12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 18 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x81 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x40 ≠ 0x40	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x81 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x40 = 0x40 *Note*: Ensure that Phase 7 registers a Vehicle Call during 1+8 Green.	Pass/Fail
23.	Set HITL Detector Input 18 = Off	
24.	DELAY .2 Seconds	
Detector 18 calls Phase 7 when Phase 7 not Green (In 2+16 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 8 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0x0	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+8.	
13.	Set HITL Detector Input 8 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 2 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x20 AND phaseStatusGroupPhaseOns.2 ≠ 0x80	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	

21.	WEND *Note:* Wait until controller reaches 2+16.	
22.	Set HITL Detector Input 2 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 18 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x80 AND phaseStatusGroupVehCalls.1 AND 0x40 ≠ 0x40	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x80 AND phaseStatusGroupVehCalls.1 AND 0x40 = 0x40 *Note:* Ensure that Phase 7 registers a Vehicle Call during 2+16 Green.	Pass/Fail
32.	Set HITL Detector Input 18 = Off	
33.	DELAY .2 Seconds	
Detector 18 calls Phase 7 when Phase 7 not Green (In 2+15 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 2 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x02 AND phaseStatusGroupPhaseOns.2 ≠ 0x40	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	

21.	WEND *Note:* Wait until controller reaches 2+15.	
22.	Set HITL Detector Input 2 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 18 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x04 AND phaseStatusGroupVehCalls.1 AND 0x40 ≠ 0x40	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x02 AND phaseStatusGroupGreens.2 = 0x40 AND phaseStatusGroupVehCalls.1 AND 0x40 = 0x40 *Note:* Ensure that Phase 7 registers a Vehicle Call during 2+15 Green.	Pass/Fail
32.	Set HITL Detector Input 18 = Off	
33.	DELAY .2 Seconds	
Detector 18 calls Phase 7 when Phase 7 not Green (In 1+13 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 3 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x01 AND phaseStatusGroupPhaseOns.2 ≠ 0x10	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	

21.	WEND *Note:* Wait until controller reaches 1+13.	
22.	Set HITL Detector Input 3 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 18 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x01 AND phaseStatusGroupGreens.2 = 0x10 AND phaseStatusGroupVehCalls.1 AND 0x40 ≠ 0x40	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x01 AND phaseStatusGroupGreens.2 = 0x10 AND phaseStatusGroupVehCalls.1 AND 0x40 = 0x40 *Note:* Ensure that Phase 7 registers a Vehicle Call during 1+13 Green.	Pass/Fail
32.	Set HITL Detector Input 18 = Off	
33.	DELAY .2 Seconds	
Detector 18 calls Phase 7 when Phase 7 not Green (In 2+5 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 2 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x12 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 2+5.	
13.	Set HITL Detector Input 2 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 18 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x12 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x40 ≠ 0x40	
19.	DELAY 1 Second	

20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x12 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x40 = 0x40 *Note*: Ensure that Phase 7 registers a Vehicle Call during 2+5 Green.	Pass/Fail
23.	Set HITL Detector Input 18 = Off	
24.	DELAY .2 Seconds	
Detector 18 calls Phase 7 when Phase 7 not Green (In 3+5 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 18 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x14 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x40 ≠ 0x40	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x14 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x40 = 0x40 *Note*: Ensure that Phase 7 registers a Vehicle Call during 3+5 Green.	Pass/Fail
23.	Set HITL Detector Input 18 = Off	
24.	DELAY .2 Seconds	
Detector 18 calls Phase 7 when Phase 7 not Green (In 4+5 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	

5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 18 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x18 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x40 ≠ 0x40	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x18 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x40 = 0x40 *Note*: Ensure that Phase 7 registers a Vehicle Call during 4+5 Green.	Pass/Fail
23.	Set HITL Detector Input 18 = Off	
24.	DELAY .2 Seconds	
Detector 18 calls Phase 7 when Phase 7 not Green (In 6+12 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	

14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 6 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x20 AND phaseStatusGroupPhaseOns.2 ≠ 0x08	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 6+12.	
22.	Set HITL Detector Input 6 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 18 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x08 AND phaseStatusGroupVehCalls.1 AND 0x40 ≠ 0x40	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x08 AND phaseStatusGroupVehCalls.1 AND 0x40 = 0x40 *Note*: Ensure that Phase 7 registers a Vehicle Call during 6+12 Green.	Pass/Fail
32.	Set HITL Detector Input 18 = Off	
33.	DELAY .2 Seconds	
Detector 18 calls Phase 7 when Phase 7 not Green (In 6+11 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	

15.	Set HITL Detector Input 6 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x20 AND phaseStatusGroupPhaseOns.2 ≠ 0x04	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 6+11.	
22.	Set HITL Detector Input 6 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 18 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x04 AND phaseStatusGroupVehCalls.1 AND 0x40 ≠ 0x40	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x20 AND phaseStatusGroupGreens.2 = 0x04 AND phaseStatusGroupVehCalls.1 AND 0x40 = 0x40 *Note*: Ensure that Phase 7 registers a Vehicle Call during 6+11 Green.	Pass/Fail
32.	Set HITL Detector Input 18 = Off	
33.	DELAY .2 Seconds	
Detector 18 calls Phase 7 when Phase 7 not Green (In 5+9 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests in green somewhere.	
6.	Set HITL Detector Input 3 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 3+5.	
13.	Set HITL Detector Input 3 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 7 = On	

16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x10 AND phaseStatusGroupPhaseOns.2 ≠ 0x01	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 5+9.	
22.	Set HITL Detector Input 7 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 18 = On	
25.	DELAY .2 Seconds	
26.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
27.	WHILE phaseStatusGroupGreens.1 = 0x10 AND phaseStatusGroupGreens.2 = 0x01 AND phaseStatusGroupVehCalls.1 AND 0x40 ≠ 0x40	
28.	DELAY 1 Second	
29.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
30.	WEND	
31.	VERIFY phaseStatusGroupGreens.1 = 0x10 AND phaseStatusGroupGreens.2 = 0x01 AND phaseStatusGroupVehCalls.1 AND 0x40 = 0x40 *Note*: Ensure that Phase 7 registers a Vehicle Call during 5+9 Green.	Pass/Fail
32.	Set HITL Detector Input 18 = Off	
33.	DELAY .2 Seconds	
Detector 18 calls Phase 7 when Phase 7 not Green (In 1+6 Green)		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 6 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+6.	
13.	Set HITL Detector Input 6 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 18 = On	
16.	DELAY .2 Seconds	

17.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
18.	WHILE phaseStatusGroupGreens.1 = 0x21 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x40 ≠ 0x40	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupGreens.1, phaseStatusGroupGreens.2, phaseStatusGroupVehCalls.1	
21.	WEND	
22.	VERIFY phaseStatusGroupGreens.1 = 0x21 AND phaseStatusGroupGreens.2 = 0x00 AND phaseStatusGroupVehCalls.1 AND 0x40= 0x40 *Note*: Ensure that Phase 7 registers a Vehicle Call during 1+6 Green.	Pass/Fail
23.	Set HITL Detector Input 18 = On	
24.	DELAY .2 Seconds	
Detector 18 extends Phase 7 until Phase 2 call exists and Detector 18 gaps and Phase 2 call continues		
1.	GET vehicleDetectorExtend.18 = [currentExtendValue]	
2.	SET vehicleDetectorExtend.18 = 40 *Note:* Set Detector 18 extend time = 4 seconds so that actuations < 4 seconds apart keep the phase extending.	
3.	GET ringStatus.1, ringStatus.2	
4.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
5.	DELAY 1 Second	
6.	GET ringStatus.1, ringStatus.2	
7.	WEND *Note:* Loop until controller rests somewhere.	
8.	Set HITL Detector Input 3 = On	
9.	DELAY .2 Seconds	
10.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
11.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
12.	DELAY 1 Second	
13.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
14.	WEND *Note:* Wait until controller reaches 3+5.	
15.	Set HITL Detector Input 3 = Off	
16.	DELAY .2 Seconds	
17.	Set HITL Detector Input 18 = On (To call and then extend Phase 7)	
18.	DELAY .2 Seconds	
19.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
20.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
21.	DELAY 1 Second	
22.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
23.	WEND *Note:* Wait until controller reaches 1+7.	

24.	Set HITL Detector Input 2 = On (Detector 18 will extend Phase 4 when there is an opposing call)	
25.	DELAY .2 Seconds *Note:* This should get Phase 7 to start timing extensions and sets up conditional logic.	
26.	GET ringStatus.2	
27.	WHILE ringStatus.2 AND 0x07 ≠ 0x01 (xxxxx001 = extension)	
28.	DELAY .1 Second	
29.	GET ringStatus.2	
30.	WEND *Note:* Wait for indication that extensions are timing. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
31.	GET phaseStatusGroupPhaseOns.1	
32.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x40= 0x40 *Note:* Verify that the extensions are on Phase 7.	Pass/Fail
33.	Set HITL Detector Input 18 = Off	
34.	DELAY 3 Seconds	
35.	Set HITL Detector Input 18 = On	
36.	DELAY .2 Seconds	
37.	Set HITL Detector Input 18 = Off	
38.	DELAY 2.8 Seconds	
39.	GET ringStatus.2	
40.	VERIFY that RESPONSE VALUE ringStatus.2 AND 0x07 = 0x01 (xxxxx001 = extension) *Note:* Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	Pass/Fail
41.	GET phaseStatusGroupPhaseOns.1	
42.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x40 = 0x40 *Note:* Verify that extensions are still timing on Phase 7 because call was entered < 4 seconds later.	Pass/Fail
43.	Set HITL Detector Input 18 = On	
44.	DELAY .2 Seconds	
45.	Set HITL Detector Input 18 = Off	
46.	DELAY 2.8 Seconds	
47.	GET ringStatus.2	
48.	VERIFY that RESPONSE VALUE ringStatus.2 AND 0x07 = 0x01 (xxxxx001 = extension) *Note:* Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	Pass/Fail
49.	GET phaseStatusGroupPhaseOns.1	
50.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x40 = 0x40 *Note:* Verify that extensions are still timing on Phase 7 because call was entered < 4 seconds later.	Pass/Fail
51.	Set HITL Detector Input 18 = On	

52.	DELAY .2 Seconds	
53.	Set HITL Detector Input 18 = Off	
54.	DELAY 4.8 Seconds *Note:* Since the time between actuations is now 5 seconds, the timer will gap and therefore disable Detector 18 from putting in any further extensions.	
55.	Set HITL Detector Input 18 = On	
56.	DELAY .2 Seconds	
57.	GET ringStatus.2	
58.	VERIFY that RESPONSE VALUE ringStatus.2 AND 0x07 ≠ 0x01 (xxxxx001 = extension) *Note:* Verify that Phase 7 is no longer timing extensions. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	Pass/Fail
59.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupVehCalls.1	
60.	WHILE (phaseStatusGroupPhaseOns.1 AND 0x40 = 0x40) AND (phaseStatusGroupVehCalls.1 AND 0x40 ≠ 0x40)	
61.	DELAY 1 Second	
62.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupVehCalls.1	
63.	WEND	
64.	VERIFY (phaseStatusGroupPhaseOns.1 AND 0x40 = 0x40) AND (phaseStatusGroupVehCalls.1 AND 0x40 = 0x40) *Note:* Ensure that Phase 7 registers a Vehicle Call.	Pass/Fail
65.	Set HITL Detector Input 2 = Off and 18 = Off	
66.	DELAY .2 Seconds	
	POST-CONDITION The Detector 18 vehicleDetectorExtend is still set to 4 seconds	
Detector 18 extends Phase 7 until Phase 2 call exists and Detector 18 gaps but not if Phase 2 call disappears		
	PRE-CONDITION The Detector 18 vehicleDetectorExtend is still set to 4 seconds	
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	
5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 4 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 4+5.	
13.	Set HITL Detector Input 4 = Off	

14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 18 = On (To call and then extend Phase 7)	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 1+7.	
22.	Set HITL Detector Input 2 = On	
23.	DELAY .2 Seconds *Note:* Detector 18 will extend Phase 7 to start timing extensions and sets up conditional logic.	
24.	GET ringStatus.1	
25.	WHILE ringStatus.2 AND 0x07 ≠ 0x01 (xxxxx001 = extension)	
26.	DELAY .1 Second	
27.	GET ringStatus.2	
28.	WEND *Note:* Wait for indication that extensions are timing. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
29.	GET phaseStatusGroupPhaseOns.1	
30.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x40 = 0x40 *Note:* Verify that the extensions are timing on Phase 7.	Pass/Fail
31.	Set HITL Detector Input 18 = Off	
32.	DELAY 3 Seconds	
33.	Set HITL Detector Input 18 = On	
34.	DELAY .2 Seconds	
35.	Set HITL Detector Input 18 = Off	
36.	DELAY 2.8 Seconds	
37.	GET ringStatus.2	
38.	VERIFY that RESPONSE VALUE ringStatus.2 AND 0x07 = 0x01 (xxxxx001 = extension) *Note:* Wait for indication that extensions are timing. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	Pass/Fail
39.	GET phaseStatusGroupPhaseOns.1	
40.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x40 = 0x40 *Note:* Verify that extensions are still timing is on Phase 7.	Pass/Fail
41.	Set HITL Detector Input 18 = On	
42.	DELAY .2 Seconds	
43.	Set HITL Detector Input 18 = Off	
44.	DELAY 2.8 Seconds	
45.	GET ringStatus.2	

46.	VERIFY that RESPONSE VALUE ringStatus.2 AND 0x07 = 0x01 (xxxxx001 = extension) *Note:* Wait for indication that extensions are timing. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	Pass/Fail
47.	GET phaseStatusGroupPhaseOns.1	
48.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x40 = 0x40 *Note:* Verify that extensions are still timing on Phase 7.	Pass/Fail
49.	Set HITL Detector Input 18 = On	
50.	DELAY .2 Seconds	
51.	Set HITL Detector Input 18 = Off, 7 = On, 2 = Off and 3 = On *Note:* Stop extending Phase 7 via Detector 18, turn on Detector 7 to keep Phase 7 extending, turn off Detector 2 opposing demand as part of the logic, and turn on Detector 3 to place opposing demand. Detector 7 will keep extending Phase 7 irrespective of Detector 18, no call on Phase 2 will reset Detector 18 gap function, and the call on Phase 3 will enable extensions to continue to time.	
52.	DELAY 4.8 Seconds *Note:* This would have the effect of allowing the extend timer to gap and therefore disable Detector 18, but since Phase 2 no longer has a call, another activation of Detector 18 will continue to extend Phase 7.	
53.	Set HITL Detector Input 18 = On and 7 = Off	
54.	DELAY .2 Seconds	
55.	Set HITL Detector Input 18 = Off	
56.	DELAY .2 Seconds	
57.	GET ringStatus.2	
58.	VERIFY that RESPONSE VALUE ringStatus.2 AND 0x07 = 0x01 (xxxxx001 = extension) *Note:* Wait for indication that extensions are timing. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	Pass/Fail
59.	GET phaseStatusGroupPhaseOns.1	
60.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.1 AND 0x40 = 0x40 *Note:* Verify that extensions are still timing on Phase 7 because even though call was > 4 seconds later, the absence of call on phase 2 allowed phase 7 to continue to time extensions.	Pass/Fail
61.	Set HITL Detector Input 3 = Off	
62.	DELAY .2 Seconds	
63.	SET vehicleDetectorExtend.18 = [currentExtendValue] *Note:* Return Detector 18 extend time back to their original value.	
Detector 18 extends Phase 15 when Phase 15 Green		
1.	GET ringStatus.1, ringStatus.2	
2.	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
3.	DELAY 1 Second	
4.	GET ringStatus.1, ringStatus.2	

5.	WEND *Note:* Loop until controller rests somewhere.	
6.	Set HITL Detector Input 7 = On	
7.	DELAY .2 Seconds	
8.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
9.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0x00	
10.	DELAY 1 Second	
11.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
12.	WEND *Note:* Wait until controller reaches 1+7.	
13.	Set HITL Detector Input 7 = Off	
14.	DELAY .2 Seconds	
15.	Set HITL Detector Input 2 = On	
16.	DELAY .2 Seconds	
17.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
18.	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x02 AND phaseStatusGroupPhaseOns.2 ≠ 0x40	
19.	DELAY 1 Second	
20.	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
21.	WEND *Note:* Wait until controller reaches 2+15.	
22.	Set HITL Detector Input 2 = Off	
23.	DELAY .2 Seconds	
24.	Set HITL Detector Input 18 = On	
25.	DELAY .2 Seconds	
26.	GET ringStatus.2	
27.	WHILE ringStatus.2 AND 0x10 ≠ 0x10 (xxx1xxxx = maxout)	
28.	DELAY 1 Second	
29.	GET ringStatus.2	
30.	WEND *Note:* Wait for Max Out Indication. Ring 1 = 2, 3, 4, 9, 11, 12, & 1 Ring 2 = 15, 16, 5, 6, 7, 8, & 13	
31.	GET phaseStatusGroupPhaseOns.2	
32.	VERIFY that RESPONSE VALUE phaseStatusGroupPhaseOns.2 AND 0x40 = 0x40 *Note:* Ensure that Max Out occurred on Phase 15.	Pass/Fail
33.	Set HITL Detector Input 18 = Off	
34.	DELAY .2 Seconds	
Teardown		
	PERFORM Detector Teardown – TC020 if not proceeding to another detector operation test case	
	Test Case Results	
Tested By:		Date Tested		

Test Case Notes:	
Version History:	v1.00 05/09/06 Initial Draft – RDR v1.01 07/03/06 Updated logic – RDR v1.02 07/05/06 Updated notes – RDR v1.03 08/17/06 Implemented script and proofed – JJ

Detector Operations Setup

Test Case: **TC019**	*Title*: Description: Constants: Variables: Pass/Fail Criteria:	**Detector Operations Setup** This procedure performs general setup of controller parameters to facilitate testing and provide consistent operation. [currentMinGrn.Phase] [currentPassage.Phase] [currentMax1.Phase] The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.	
Test Step Number	*Test Procedure*		*Results*
Setup			
1.	CONFIGURE the controller for: 1. 4-phase diamond operation with Phase 3 and 7 in sequence 2. Connection to TS 2 Tester Box (BIU's enabled) 3. Vehicle and Pedestrian Recalls are all off 4. Default data loaded 5. Call non-locking memory is set so that calls are not remembered if a call is removed 6. Yellow clearances are set to at least 3.5 seconds 6. Red clearances are set to at least 1.5 seconds *Note:* The setup for different manufacturer's controllers will be different. Reference should be made to a document containing such information. Eagle Configuration Notes: Unit Data – Startup & Misc – Alt Sequence = 16 Unit Data – Port 1 Data = Enable T&F 1=4, DET 1-4, and Malfunction Unit Phase Data – Initialization & N.A. Response Phase 4 and 7 = Change "Dark" TO "Inactive" Naztec Configuration Notes: Econolite Notes:		
2.	FOR Phase = 1 TO 16		
3.	GET phaseMinimumGreen.Phase, phasePassage.Phase, and phaseMaximum1.Phase		
4.	RECORD RESPONSE VALUE in [currentMinGrn.Phase], [currentPassage.Phase] and [currentMax1.Phase] *Note:* These values will be restored at the end of the test case.		

5.	SET phaseMinimumGreen.Phase = 5, phasePassage.Phase = 0, and phaseMaximum1.Phase = 25 *Note:* min = 5 second, passage = 4 seconds, and max = 25 seconds.			
6.	NEXT Phase			
Test Case Results				
Tested By:		Date Tested		Pass/Fail
Test Case Notes:	<notes>			
Version History:	v1.00 05/05/06 Initial Draft – RDR v1.01 06/27/06 Updated test values – RDR v1.02 07/05/06 Updated notes – RDR v1.03 07/27/06 Implemented script and proofed – JJ			

Detector Operations Teardown

Test Case: **TC020**	*Title*: Description: Constants: Variables: Pass/Fail Criteria:	**Detector Operations Teardown** This procedure restores original controller parameters after executing Detector Operation Setup – TC001 The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.		
Test Step Number	*Test Procedure*			*Results*
Teardown				
1.	FOR Phase = 1 TO 16			
2.	SET phaseMinimumGreen.Phase = [currentMinGrn.Phase], phasePassage.Phase = [currentPassage.Phase], phaseMaximum1.Phase = [currentMax1.Phase]			Pass/Fail
3.	NEXT Phase			
Test Case Results				
Tested By:		Date Tested		Pass/Fail
Test Case Notes:	<notes>			
Version History:	v1.00 05/08/06 Initial Draft – RDR v1.01 07/27/06 Implemented script and proofed – JJ			

Detector Delay

Test Case: **TC021**	*Title*: Description: Constants: Variables: Pass/Fail Criteria:	**Detector Delay** Verifies that, when programmed, Detectors 2, 3, 4, 6, 7, and 8 delay entering a call for the parent phase when the parent phase is red. currentDetectorDelay.2 (.4, 6, 7, and 8) The DUT shall pass every verification step included within the Test Case in order to pass the Test Case.

Test Step Number	*Test Procedure*	*Results*
Setup		
	FOR DetectorNumber = 2, 3, 4, 6, 7, and 8	
	GET vehicleDetectorDelay.DetectorNumber	
	RECORD RESPONSE VALUE in [currentDetectorDelay.DetectorNumber] *Note:* These values will be restored at the end of the test case.	
	SET vehicleDetectorDelay.DetectorNumber = 2	Pass/Fail
	NEXT Phase	
Detector 2 Delay		
	GET ringStatus.1, ringStatus.2	
	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
	DELAY 1 Second	
	GET ringStatus.1, ringStatus.2	
	WEND *Note:* Loop until controller rests in green somewhere.	
	Set phaseControlGroupVehCall.1 = 0x20	
	DELAY .2 Seconds	
	Set phaseControlGroupVehCall.1 = 0x00	
	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x21 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
	DELAY 1 Second	
	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
	WEND *Note:* Wait until controller reaches 1+6.	
	Set HITL Detector Input 2 = On	
	DELAY 1.9 Seconds	
	GET phaseStatusGroupVehCalls.1	
	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 = 0x00 *Note:* Verify that detector call is not active yet.	Pass/Fail
	DELAY .2 Seconds	
	GET phaseStatusGroupVehCalls.1	
	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 = 0x02 *Note:* Verify that detector call is now active.	Pass/Fail
	Set HITL Detector Input 2 = Off	

Detector 3 Delay		
	GET ringStatus.1, ringStatus.2	
	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
	DELAY 1 Second	
	GET ringStatus.1, ringStatus.2	
	WEND *Note:* Loop until controller rests in green somewhere.	
	Set phaseControlGroupVehCall.1 = 0x40	
	DELAY .2 Seconds	
	Set phaseControlGroupVehCall.1 = 0x00	
	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x41 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
	DELAY 1 Second	
	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
	WEND *Note:* Wait until controller reaches 1+7.	
	Set HITL Detector Input 3 = On	
	DELAY 1.9 Seconds	
	GET phaseStatusGroupVehCalls.1	
	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 = 0x00 *Note:* Verify that detector call is not active yet.	Pass/Fail
	DELAY .2 Seconds	
	GET phaseStatusGroupVehCalls.1	
	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 = 0x04 *Note:* Verify that detector call is now active.	Pass/Fail
	Set HITL Detector Input 3 = Off	
Detector 4 Delay		
	GET ringStatus.1, ringStatus.2	
	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
	DELAY 1 Second	
	GET ringStatus.1, ringStatus.2	
	WEND *Note:* Loop until controller rests in green somewhere.	
	Set phaseControlGroupVehCall.1 = 0x80	
	DELAY .2 Seconds	
	Set phaseControlGroupVehCall.1 = 0x00	
	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	

	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x81 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
	DELAY 1 Second	
	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
	WEND *Note:* Wait until controller reaches 1+8.	
	Set HITL Detector Input 4 = On	
	DELAY 1.9 Seconds	
	GET phaseStatusGroupVehCalls.1	
	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 = 0x00 *Note:* Verify that detector call is not active yet.	Pass/Fail
	DELAY .2 Seconds	
	GET phaseStatusGroupVehCalls.1	
	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 = 0x08 *Note:* Verify that detector call is now active.	Pass/Fail
	Set HITL Detector Input 4 = Off	
Detector 6 Delay		
	GET ringStatus.1, ringStatus.2	
	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
	DELAY 1 Second	
	GET ringStatus.1, ringStatus.2	
	WEND *Note:* Loop until controller rests in green somewhere.	
	Set phaseControlGroupVehCall.1 = 0x02	
	DELAY .2 Seconds	
	Set phaseControlGroupVehCall.1 = 0x00	
	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x12 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
	DELAY 1 Second	
	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
	WEND *Note:* Wait until controller reaches 2+5.	
	Set HITL Detector Input 6 = On	
	DELAY 1.9 Seconds	
	GET phaseStatusGroupVehCalls.1	

	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 = 0x00 *Note:* Verify that detector call is not active yet.	Pass/Fail
	DELAY .2 Seconds	
	GET phaseStatusGroupVehCalls.1	
	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 = 0x20 *Note:* Verify that detector call is now active.	Pass/Fail
	Set HITL Detector Input 6 = Off	
Detector 7 Delay		
	GET ringStatus.1, ringStatus.2	
	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
	DELAY 1 Second	
	GET ringStatus.1, ringStatus.2	
	WEND *Note:* Loop until controller rests in green somewhere.	
	Set phaseControlGroupVehCall.1 = 0x04	
	DELAY .2 Seconds	
	Set phaseControlGroupVehCall.1 = 0x00	
	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x14 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
	DELAY 1 Second	
	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
	WEND *Note:* Wait until controller reaches 3+5.	
	Set HITL Detector Input 7 = On	
	DELAY 1.9 Seconds	
	GET phaseStatusGroupVehCalls.1	
	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 = 0x00 *Note:* Verify that detector call is not active yet.	Pass/Fail
	DELAY .2 Seconds	
	GET phaseStatusGroupVehCalls.1	
	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 = 0x40 *Note:* Verify that detector call is now active.	Pass/Fail
	Set HITL Detector Input 7 = Off	

Detector 8 Delay		
	GET ringStatus.1, ringStatus.2	
	WHILE ringStatus.1 ≠ 0x03 AND ringStatus.2 ≠ 0x03	
	DELAY 1 Second	
	GET ringStatus.1, ringStatus.2	
	WEND *Note:* Loop until controller rests in green somewhere.	
	Set phaseControlGroupVehCall.1 = 0x04	
	DELAY .2 Seconds	
	Set phaseControlGroupVehCall.1 = 0x00	
	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
	WHILE phaseStatusGroupPhaseOns.1 ≠ 0x18 AND phaseStatusGroupPhaseOns.2 ≠ 0X00	
	DELAY 1 Second	
	GET phaseStatusGroupPhaseOns.1, phaseStatusGroupPhaseOns.2	
	WEND *Note:* Wait until controller reaches 4+5.	
	Set HITL Detector Input 8 = On	
	DELAY 1.9 Seconds	
	GET phaseStatusGroupVehCalls.1	
	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 = 0x00 *Note:* Verify that detector call is not active yet.	Pass/Fail
	DELAY .2 Seconds	
	GET phaseStatusGroupVehCalls.1	
	VERIFY that RESPONSE VALUE phaseStatusGroupVehCalls.1 = 0x80 *Note:* Verify that detector call is now active.	Pass/Fail
	Set HITL Detector Input 8 = Off	
Teardown		
	FOR DetectorNumber = 2, 3, 4, 6, 7, and 8	
	SET vehicleDetectorDelay.DetectorNumber = [currentDetectorDelay.DetectorNumber] *Note:* Restore the original value.	Pass/Fail
	NEXT Phase	

Test Case Results				
Tested By:		Date Tested		Pass/Fail
Test Case Notes:	<notes>			
Version History:	v1.00 06/11/06 Initial Draft – RDR v1.01 07/05/06 Updated notes – RDR			

REFERENCES FOR APPENDIX G

1. NTCIP Laboratory Testing for Actuated Signal Controllers, Summary Report for ASSHTO Project 475070. Published by Texas Transportation Institute. http://tti.tamu.edu/documents/TTI-2006-1.pdf. Accessed June 7, 2006.

2. NTCIP 1202 – Object Definitions for Actuated Traffic Signal Controller Units, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=1202. Accessed June 21, 2006.

3. Balke, K., R. Engelbrecht, S. Sunkari, J. Charara, TTI'S Hardware-In-The-Loop Traffic Signal Controller Evaluation System. Published by Texas Transportation Institute. http://tti.tamu.edu/documents/5-1752-01-1.pdf. Accessed August 23, 2006.

4. DMS-11170, Fully Actuated, Solid-State Traffic Signal Controller Assembly, Departmental Material Specifications 7-115, Section 19. Published by TxDOT. http://manuals.dot.state.tx.us/dynaweb/colmates/dms/@ebt-link;?target=idmatch(s070019). Accessed July 29, 2005.

5. NTCIP 1201 – Global Object Definitions. A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=1201. Accessed July 25, 2005.

6. NTCIP 8007 – Testing and Conformity Assessment Documentation within NTCIP Standards Publications, A Joint Publication of AASHTO, ITE, and NEMA. http://www.ntcip.org/library/standards/default.asp?documents=yes&qreport=no&standard=8007. Accessed July 25, 2005.